INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

GENERACIÓN DE COMPORTAMIENTOS COLABORATIVOS UTILIZANDO APRENDIZAJE POR REFUERZO EN UN MEDIO DE AGENTES DE FÚTBOL.

TESIS QUE PARA OPTAR EL GRADO DE
MAESTRO EN CIENCIAS COMPUTACIONALES
PRESENTA

CARLOS REYES RICO

Asesor:	Dra. MA. DE LOS ÁNGELES JUNCO REY
Asesor Externo:	Dr. EDGAR EMMANUEL VALLEJO CLEMENTE

Comité de tesis:	Dr. EDGAR EMMANUEL VALLEJO CLEMENTE,	Presidente
	Dr. VÍCTOR MANUEL DE LA CUEVA HERNÁNDEZ,	Secretario
	Dra. MA. DE LOS ÁNGELES JUNCO REY,	Vocal
	Dr. GILDARDO SÁNCHEZ ANTE,	Vocal

Atizapán de Zaragoza, Edo. Méx., Enero de 2008.

DEDICACIONES Y RECONOCIMIENTOS

Esta disertación es dedicada a mi familia Herlinda Rico Albarrán, Carlos Reyes Aguilar y Mario Reyes Rico por el invaluable respeto y admiración que les tengo, que ha servido como aliciente para realizar este trabajo.

Mi más sincera gratitud para mi asesor de investigación, la Dra. María de los Ángeles Junco Rey por su generoso tiempo y encaminamiento correcto de esta investigación; sin los cuales este trabajo no hubiese sido posible.

También quiero expresar mi gratitud al Proyecto de Robots Cuadrúpedos de la División de Investigación en el Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Estado de México; especialmente a todos los profesores que forman parte del Proyecto de Robots Cuadrúpedos por apoyar esta investigación.

Finalmente, agradezco a mi familia por su cariño, paciencia, soporte y estímulo a lo largo de mi vida.

RESUMEN

La generación de comportamientos para agentes en un medio dinámico representa una valiosa aportación pues se puede obtener ciertas metas establecidas por las ciencias computacionales en el campo de la inteligencia artificial. Algunas de esas metas son: diseñar nuevas metodologías computacionales para la solución de problemas, la dirección de análisis, diseño e implantación de proyectos de desarrollo tecnológico computacional capaz de generar nuevo conocimiento, entre otras.

La inteligencia artificial se puede ver como el estudio de métodos que se pueden construir por medio de agentes inteligentes. Los agentes inteligentes tienen como propiedades fundamentales: inteligencia, autonomía, habilidad de aprender, cooperación, entre otros. La habilidad para aprender, es la característica de los agentes de interpretar nueva información del ambiente y de esta manera éstos pueden generar acciones o comportamientos nuevos gracias al uso de dicha información.

El aprendizaje permite a los agentes mejorar su comportamiento al realizar tareas muy específicas a través del tiempo. El agente es capaz de aprender de su propia experiencia al ejecutar una acción y de esta manera no cometer los mismos errores en el futuro. La manera en la cual se intenta simular el actuar humano para la generación del aprendizaje por refuerzo es conocido como el aprendizaje básico humano pues genera comportamientos para brindar soluciones a los agentes[1].

Esta investigación se enfoca esencialmente al análisis, desarrollo e implementación de una herramienta útil para los agentes y éstos puedan generar comportamientos de manera autónoma. De manera específica se utiliza la herramienta *XML Behaviour Control*[2] para generar una nueva solución de generación de comportamientos. La integración con esta herramienta resulta bastante útil pues su arquitectura computacionalmente es estable y sobre todo modular, esto es, resulta sencillo generar nuevas soluciones a problemas de decisión de agentes en este simulador.

El nombre de la nueva herramienta es BADRL, *Behaviour Agent Definition by Reinforcement Learning*, el cual será modular para cualquier tipo de ambiente. Se define un sistema como modular en el instante en que será sencillo cambiar de solución para que los agentes puedan aprender las acciones que estén realizando sin importar lo que represente esa acción, pues no es lo mismo caminar hacia una pelota con cierta velocidad para jugar fútbol que caminar de manera dirigida para salir de un laberinto. El uso de algoritmos de inteligencia artificial representa un gran reto y una gran motivación para utilizar agentes que puedan crear comportamientos de manera autónoma. La finalidad de esta tesis pretende demostrar que el algoritmo Q-Learning resulta bastante útil para la generación de comportamientos para agentes que juegan fútbol.

[1] Sutton, et al. *Reinforcement Learning. An Introduction.* p. 56-60.
[2] Vega, et al. "Major behavior definition of football agents through XML" , p. 4.

CONTENIDO DE LA TESIS

ÍNDICE DE FIGURAS. 7
ÍNDICE DE TABLAS. 10
I. MOTIVACIÓN Y RAZONES DE ESTUDIO SOBRE APRENDIZAJE POR REFUERZO. 11
II. ANTECEDENTES Y ESTADO DEL ARTE ACERCA DEL TRABAJO DE INVESTIGACIÓN. 13
1. INTRODUCCIÓN DE LOS ANTECEDENTES DEL TRABAJO DE INVESTIGACIÓN.13
2. TRABAJO DE INVESTIGACIÓN Y AVANCES DEL PROYECTO ROBOTS CUADRÚPEDOS..... 14
A. INICIOS E HISTORIA DEL EQUIPO TECRAMS. 14
B. TRABAJOS E INVESTIGACION DEL PROYECTO. 15
C. RESULTADOS Y TRABAJO FUTURO DEL EQUIPO TECRAMS. 18
3. DEFINICIONES REFERENTES DEL APRENDIZAJE POR REFUERZO. 21
A. POLITICAS. 23
B. FUNCIÓN DE RECOMPENSA. 23
C. FUNCIÓN DE VALOR. 24
D. MODELO DEL AMBIENTE. 24
E. RETROALIMENTACIÓN POR EVALUACIÓN. 25
4. TRABAJO SOBRE APRENDIZAJE POR REFUERZO EN LA ORGANIZACIÓN *ROBOCUP*. 25
A. TRABAJOS TEÓRICOS SOBRE APRENDIZAJE POR REFUERZO. 26
B. INVESTIGACIONES SOBRE COMPORTAMIENTOS POR MEDIO DEL USO DE APRENDIZAJE POR REFUERZO EN LA LIGA DE SIMULACIÓN DE *ROBOCUP*. ... 29
III. PLANTEAMIENTO DEL PROBLEMA Y SU JUSTIFICACIÓN. 34
1. INTRODUCCIÓN AL TRABAJO Y HERRAMIENTAS UTILIZADAS POR TECRAMS. 34
2. ANÁLISIS Y PROBLEMAS DE LOS APOYOS UTILIZADOS EN EL PROYECTO ROBOTS CUADRÚPEDOS. 36
A. LA HERRAMIENTA XABSL. 36
B. LA HERRAMIENTA *XML BEHAVIOUR CONTROL*. 36
C. LA HERRAMIENTA ABG. 37
D. SOLUCIONES TEMPORALES QUE SE UTILIZA EN EL PROYECTO DE LOS ROBOTS CUADRÚPEDOS. 38
3. PLANTEAMINETO DEL PROBLEMA DE LA GENERACIÓN DE COMPORTAMIENTOS PARA TECRAMS..... 39
4. JUSTIFICACIÓN DEL USO DE APRENDIZAJE POR REFUERZO PARA LA GENERACIÓN DE COMPORTAMIENTOS. 40
A. PROGRAMACIÓN DINÁMICA. 40
B. MÉTODOS MONTE CARLO. 40
C. MÉTODOS DE DIFERNCIA TEMPORAL. 41
IV. *BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING* COMO HERRAMIENTA PARA LA GENERACIÓN DE COMPORTAMIENTOS DE AGENTES. 42

1. INTRODUCCIÓN A LA HERRAMIENTA BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING (*BADRL*). 42
2. CREACIÓN DE UNA NUEVA SOLUCIÓN EN EL SIMULADOR *XML BEHAVIOUR CONTROL*. 43
A. IMPLEMENTACIÓN DE BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING. 43
B. GENERACIÓN DEL REPOSITORIO DE ACCIONES PARA *BADRL*. 45
C. LECTURA DE PARÁMETROS PARA Q-LEARNING GRACIAS AL ARCHIVO DE CONFIGURACIÓN DE *BADRL*. 46
D. DEFINICIÓN DE METAS Y OBJETIVOS PARA LA GENERACIÓN DE COMPORTAMIENTOS EN *BADRL*. 47
E. DEFINICIÓN DE POLÍTICAS Y LA FUNCIÓN DE RETROALIMENTACIÓN PARA BADRL. 48
3. PROCESO DE GENERACIÓN DE COMPORTAMIENTOS QUE USA BADRL. 49
A. GENERACIÓN DEL AMBIENTE SIMULADO PARA EL *XML BEHAVIOUR CONTROL*. 51
B. SELECCIÓN DE ACCIONES POR MEDIO DE Q-LEARNING. 51
C. USO DE LAS METAS Y OBJETIVOS PARA GENERAR COMPORTAMIENTOS. . 54
D. EJECUCIÓN DE LOS CRITERIOS DE EVALUACIÓN Y RETROALIMENTACIÓN PARA Q-LEARNING. 55
E. GENERACIÓN DE REPORTE ESTADÍSTICO CON LA INFORMACIÓN ARROJADA POR BADRL. 56
V. RESULTADOS Y EVALUACIONES DE LOS COMPORTAMIENTOS GENERADOS POR *BADRL*. 58
1. INTRODUCCIÓN A LAS EVALUACIONES DE LOS COMPORTAMIENTOS. 58
2. EVALUACIÓN DEL COMPORTAMIENTO “CONDUCCIÓN”. 61
3. EVALUACIÓN DEL COMPORTAMIENTO “BÚSQUEDA DE LA BOLA”. 65
4. EVALUACIÓN DEL COMPORTAMIENTO “APROXIMACIÓN DE LA BOLA”. 69
5. EVALUACIÓN DEL COMPORTAMIENTO “DISPARO A PORTERÍA”. 72
6. EVALUACIÓN DEL COMPORTAMIENTO “TRIANGULACIÓN PARA DISPARO A PORTERÍA”. 74
7. ANÁLISIS Y CONCLUSIONES DE LOS COMPORTAMIENTOS GENERADOS POR *BADRL*. 75
A. RESULTADOS PARA EL COMPORTAMIENTO BÚSQUEDA DE LA BOLA. 75
B. RESULTADOS PARA EL COMPORTAMIENTO CONDUCCIÓN. 75
C. RESULTADOS PARA EL COMPORTAMIENTO DISPARO A PORTERÍA. 78
D. RESULTADOS PARA EL COMPORTAMIENTO SEGUIR LA BOLA. 79
E. RESULTADOS PARA EL COMPORTAMIENTO TRIANGULACIÓN PARA DISPARO A PORTERÍA. 80
F. RESULTADOS PARA EL COMPORTAMIENTO EVADIR OBSTÁCULOS. 82
VI. CONCLUSIONES FINALES Y TRABAJO FUTURO DE LA TESIS. 84
1. CAPACIDAD DE MANEJARSE EN CUALQUIER TIPO DE PLATAFORMA QUE PUEDA CONECTARSE CON JAVA5. 84
2. CAPACIDAD DE MANEJAR UN NÚMERO ILIMITADO DE ACCIONES. 84
3. CAPACIDAD DE MANEJAR UN NÚMERO ILIMITADO DE REGLAS DE EVALUACIÓN. 85
4. CAPACIDAD DE UTILIZARSE EN CUALQUIER TIPO DE AMBIENTE: INTERNO O EXTERNO. 85

5. CAPACIDAD DE CAMBIAR PARÁMETROS DE CONFIGURACIÓN DEL ALGORTIMO Q-LEARNING DE MANERA SIMPLE Y SENCILLA. 85
6. TRABAJO FUTURO DE LA INVESTIGACIÓN. 86
A. GENERAR COMPORTAMIENTOS EN DONDE VARIOS JUGADORES APRENDAN UN COMPORTAMIENTO COLABORATIVO AL MISMO TIEMPO..... 87
B. GENERAR COMPORTAMIENTOS POR APRENDIZAJE POR REFUERZO UTILIZANDO OTRO ALGORITMO COMO SARSA Y ENCONTRAR PATRONES PARA MEJORAR BADRL. 87
C. REALIZAR PRUEBAS DE BADRL CON OTRAS PLATAFORMAS DE ROBOTS FÍSICOS COMO POR EJEMPLO: PIONNER 3ATX, 3ADX Y AMIGOBOTS. 88
VIII. ANEXOS. 93
1. EJEMPLO DEL LENGUAJE QUE UTILIZA *XABSL.* 93
2. EJEMPLO DEL LENGUAJE QUE UTILIZA *XML BEHAVIOUR CONTROL.* 94
3. EJEMPLO DE CONFIGURACIÓN DEL ARCHIVO *conf.xml.* 95
4. EJEMPLO DE CONFIGURACIÓN DEL ARCHIVO *actions.xml.* 96
5. GENERACIÓN DE NUEVOS COMPORTAMIENTOS PARA EL PROYECTO *robocup3* EN EL SIMULADOR *XML BEHAVIOUR CONTROL.* 97
6. INSTALACIÓN DE LA APLICACIÓN. 98
A. INSTALACIÓN DEL AMBIENTE GRÁFICO DE PROGRAMACIÓN, ECLIPSE 98
1. Bajar el ambiente gráfico de la página principal de Eclipse. 98
2. Instalación del ambiente en la computadora personal del programador. 98
3. Abrir y ejecutar el programa Eclipse 100
4. Crear el espacio de trabajo (workspace) en Eclipse para instalar XML Behaviour Control y BADLR. 100
B. INSTALACIÓN DEL XML BEHAVIOUR CONTROL Y BADRL. 101
5. Descomprimir archivo robocup4.zip 101
6. Copiar folder robocup4 al workspace creado 101
7. Cargar robocup4 como workspace en Eclipse 103
8. Cargar robocup4 del workspace 103
9. Buscar workspace en el directorio de Windows 103
10. Cerrar pantalla Welcome 105
11. Abrir código robocup2 105
12. Abrir robocup2/rl.files_conf la clase FileConf.java; Modificar Files.CONF_FILE, Files.ACTIONS_FILE, Files.DTD_FILE y Files.RULES_FILE 106
13. Abrir robocup2/rl y dar clic botón derecho sobre MainRLClass.java 106
14. Disfrutar el comportamiento y empezar la investigación 106
7. MANUAL DE USUARIO. 108
A. CREAR AMBIENTE DE PRUEBAS 108
B. ANALIZAR EL COMPORTAMIENTO DESEADO. 109
C. PROGRAMAR POSIBLES COMPORTAMIENTOS 109
1. Generar pool de acciones posibles que puede realizar el agente 110
2. Generar reglas para evaluación 112
3. Probar acciones y reglas corriendo Q-Learning 113
4. Analizar reporte final 113
5. Obtener comportamiento final aprendido 113
6. Cambiar parámetros de configuración para el uso de Q-Learning 114
7. Volver a la fase de pruebas 115

ÍNDICE DE FIGURAS.

Figura 1. El avance tecnológico representado en un robot Sony AIBO®..... 11
Figura 2. Robot Sony AIBO Modelo 210 con el control del esférico en el terreno de fútbol..... 15
Figura 3. Arquitectura de código actual de TecRams. 16
Figura 4. Organización de módulos en el proyecto TecRams. 16
Figura 5. Organización de módulos en el proyecto GermanTeam. 17
Figura 6. Terreno de juego utilizado por los Sony Aibo para jugar fútbol..... 19
Figura 7. Interfaz gráfica del XML Behaviour Control.. 20
Figura 8. Cromosoma de una generación representando un comportamiento de buscar la pelota.20
Figura 9. El objetivo principal del aprendizaje por refuerzo. 22
Figura 10. Recompensa por política o regla en el sistema BADRL. 24
Figura 11. Comparativo un equipo que utiliza aprendizaje por refuerzo frente a uno aleatorio. .. 27
Figura 12. Equipos compuestos por 11 jugadores en el campo de simulación. 27
Figura 13. Resultados comparativos entre PIPE, COPIPE y TD-Q para el comportamiento de pase. 28
Figura 14. Instancias principales utilizadas en la arquitectura Clay..... 29
Figura 15. Ambiente ideal para un tiro a portería..... 29
Figura 16. Agente K1 conservando el esférico sin que se intercepte por los contrarios. 30
Figura 17. Comparativo de aprendizaje entre Sarsa y Q-Learning. 31
Figura 18. Flujo de tareas para representar tres diferentes roles en un juego de fútbol. 31
Figura 19. Agentes simulados que realizan pases y tiros de acuerdo a roles programados. 32
Figura 20. Ángulo de visión de un *humanoide* para obtener la recompensa de sus acciones. 33
Figura 21. Control de estados por medio del XML Behavior Control 34
Figura 22. Proceso de generación de comportamientos utilizando el simulador XML Behaviour Control y el XABSL. 35
Figura 23. Generación de dos cromosomas con un comportamiento cada uno 37
Figura 24. Proceso de generación de comportamientos utilizando el simulador XML Behaviour Control y el XABSL. 39
Figura 25. Esquema de la investigación utilizando aprendizaje por refuerzo con las actuales herramientas de *TecRams.* 42
Figura 26. Esquema de la solución por agregar en el XML Behaviour Control.. 43
Figura 27. Procesos generales de *BADRL.* 44
Figura 28. Relación de clases dentro de *Free Connectionist Q-Learning Java Framework.* 44
Figura 29. Estado que manda a llamar la acción atómica de caminar lento y una transición al primer estado que lo mandó a llamar..... 45
Figura 30. Generación de transiciones del primer estado por BADRL a los estados con acciones atómicas. 45
Figura 31. Parámetro alfa como tamaño del paso en la fórmula general de Q-Learning. 46
Figura 32. Uso del archivo *conf.xml* para modificar los variables utilizadas por Q-Learning. 46
Figura 33. Definición de la meta por medio de la primera regla de *rules.xml* para la conducción de la pelota. 47
Figura 34. Criterio de evaluación para determinar si un agente ha logrado aprender la conducción de la bola. 48

Figura 35. Archivo *rules.xml* con la descripción de las primeras 3 reglas para la conducción de la pelota. 49
Figura 36. Proceso general de la generación de comportamientos de *BADRL.* 50
Figura 37. Ambiente especializado para una evaluación correcta del comportamiento *approachToBall.* 51
Figura 38. Código de la función principal de Q-Learning para seleccionar una acción. 52
Figura 39. Código del método público count() de Q-Learning para seleccionar una acción. 53
Figura 40. Código del método privado *updateWeights()* de Q-Learning para seleccionar una acción. 54
Figura 41. Recompensas para los criterios de evaluación para BADRL. 55
Figura 42. Recompensas para los criterios de evaluación para BADRL. 55
Figura 43. Forma de activar las banderas para retroalimentar a Q-Learning con las recompensas 56
Figura 44. Reporte final generado por BADRL para la creación del comportamiento de la conducción. 57
Figura 45. Tiempo inicial y final del archivo HTML con los resultados finales de BADRL. 59
Figura 46. Tiempo inicial y final del archivo HTML con los resultados finales de BADRL. 59
Figura 47. Cuadro principal del reporte generado por BADRL. 60
Figura 48. Conducción del balón por parte del agente de un punto A a un punto B. 62
Figura 49. Comportamiento de conducción aprendido por *BADRL* con sólo 2 transiciones a estados diferentes. 64
Figura 50. Comportamiento de conducción creado manualmente con 4 transiciones a estados diferentes 65
Figura 51. Tamaño y posición de la cámara de color localizada dentro del robot AIBO modelo 210. 66
Figura 52. Alineación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control* 68
Figura 53. Comportamiento de búsqueda la bola aprendido por *BADRL* con sólo 1 transició a estado diferente. 69
Figura 54. Comportamiento de búsqueda de la bola creado manualmente con 4 transiciones a estados diferentes. 69
Figura 55. Aproximación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control.* 71
Figura 56. Comportamiento de acercarse a la bola creado manualmente con 7 transiciones a estados diferentes 72
Figura 57. Aproximación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control.* 73
Figura 58. Comportamiento de disparar a la portería creado por BADRL con 2 transiciones a estados diferentes. 73
Figura 59. Ambiente de pruebas para la triangulación antes del disparo a portería. 75
Figura 60. Comportamiento de mandar, recibir la bola y disparar a la portería creado por BADRL con 4 transiciones a estados diferentes. 74
Figura 61. Definición matemática del algoritmo Q-Learning. 86
Figura 62. Uso del archivo *conf.xml* para modificar los variables utilizadas por Q-Learning. 86
Figura 63. Partes y módulos principales del desarrollo interno de la herramienta *BADRL.* 89
Figura 64. Sitio principal para bajar el programa de ambiente gráfico: Eclipse. 99
Figura 65. Ejemplo para descomprimir Eclipse en el ordenador. 99
Figura 66. Selección del ícono eclipse.exe dentro de la carpeta de Eclipse. 100
Figura 67. Ventana principal para definir el espacio de trabajo en Eclipse. 101

Figura 68. Descomprimir el archivo robocup4 con el simulador XML Behaviour Control. 102
Figura 69. Ejemplo para copiar el folder que contiene el código del simulador y BADRL al espacio de trabajo por default creado. 102
Figura 70. Menú del programa Eclipse para importar el código del espacio de trabajo. 103
Figura 71. Ejemplo para importar el código del espacio de trabajo. 104
Figura 72. Ejemplo para buscar la carpeta robocup4 en el espacio de trabajo en Eclipse. 104
Figura 73. Visualización de los archivos y organización del código del XML Behaviour Control. 105
Figura 74. Ejemplo para cambiar la referencia de todos los archivos de BADRL..... 106
Figura 75. Ejemplo gráfico para correr el programa BADRL en Eclipse. 107
Figura 76. Corriendo BADRL en el simulador XML Behaviour Control. 107
Figura 77. Ambiente especializado para evaluación correcta del comportamiento approachToBall. 108
Figura 78. Comportamiento passHelper, recibe la pelota y es capaz de arrojarla a un punto aproximado. 110
Figura 79. Conjunto de acciones posibles del agente por medio del archivo actions.xml. 112
Figura 80. Conjunto de reglas posibles para evaluar a un agente para un comportamiento específico. 113
Figura 81. Ejemplo visual del reporte final que arroja BADRL después de aprender un comportamiento 114
Figura 82. Archivo conf.xml que sirve para modificar los parámetros de configuración de Q-learning de BADRL 114

ÍNDICE DE TABLAS.

Tabla 1. Probabilidades de Mutación. 38
Tabla 2. Abreviaturas para la evaluación de comportamientos. 61
Tabla 3. Parámetros definidos para el comportamiento conducción. 63
Tabla 4. Parámetros definidos para el comportamiento búsqueda de la bola. 67
Tabla 5. Parámetros definidos para el comportamiento aproximación de la bola. 70
Tabla 6. Parámetros definidos para el comportamiento disparo a portería. 72
Tabla 7. Parámetros definidos para el comportamiento triangulación para disparo a portería. 74
Tabla 8. Resultados de BADRL para el comportamiento de alineación. 76
Tabla 9. Relación de número y nombres de acciones de la tabla 8. 76
Tabla 10. Resultados de BADRL para el comportamiento de conducción. 77
Tabla 11. Relación de número y nombres de acciones de la tabla 10. 77
Tabla 12. Porcentaje de efectividad de BADRL para el comportamiento conducción. 78
Tabla 13. Resultados de BADRL para el comportamiento disparo a portería. 78
Tabla 14. Relación de número y nombres de acciones de la tabla 13. 79
Tabla 15. Porcentaje de efectividad de BADRL para el comportamiento disparo a portería. 79
Tabla 16. Resultados de BADRL para el comportamiento seguir la bola. 79
Tabla 17. Relación de número y nombres de acciones de la tabla 16. 80
Tabla 18. Porcentaje de efectividad de BADRL para el comportamiento seguir la bola. 80
Tabla 19. Resultados de BADRL para el comportamiento triangulación para disparo a portería. 81
Tabla 20. Relación de número y nombres de acciones de la tabla 19. 81
Tabla 21. Porcentaje de efectividad de BADRL para el comportamiento triangulación para disparo a portería. 81
Tabla 22. Resultados de BADRL para el comportamiento evadir obstáculos. 82
Tabla 23. Relación de número y nombres de acciones de la tabla 22. 82
Tabla 24. Porcentaje de efectividad de BADRL para el comportamiento evadir obstáculos. 82

I. MOTIVACIÓN Y RAZONES DE ESTUDIO SOBRE APRENDIZAJE POR REFUERZO.

Las etapas de la historia se generan por cambios radicales en el ambiente humano. A partir de la segunda guerra mundial se generaron agentes capaces de cambiar la vida humana. Los robots y el hombre empezaron a seguir un camino hacia una nueva etapa para la generación de soluciones en los ámbitos sociales, culturales, políticos y económicos. El avance exponencial tecnológico es resultado de las ideas revolucionarias en robótica con el hombre.

Los avances tecnológicos representan cambios significativos en las herramientas electrónicas utilizadas por los robots. Los agentes computacionales cada vez cuentan con más y mejores características para interactuar con el medio y las personas. Los robots *Sony AIBO*®[3] son un ejemplo del avance tecnológico actual puesto que contienen sensores, servo motores, tarjetas inalámbricas, bocinas, micrófonos, cámaras, entre otros componentes [Figura 1].

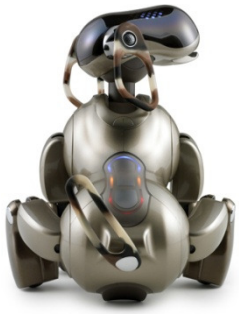


Figura 1. El avance tecnológico representado en un robot Sony AIBO®

La manipulación de robots físicos representa un gran reto para generar soluciones en ambientes dinámicos. La inteligencia artificial es ahora un enfoque para dotar autonomía a un agente para diferentes tipos de ambientes[4]. Desde hace una década, las investigaciones y discusiones a nivel mundial se han ido interesando especialmente en agentes autónomos que tengan un desempeño muy similar al comportamiento humano[5]. El motor fundamental de la inteligencia artificial es conocido como aprendizaje. La integración de cualquier tipo de aprendizaje a un robot para generar soluciones a la vida humana es parte esencial de la etapa robótica actual.

La generación de comportamientos para agentes en un medio dinámico representa una valiosa aportación para obtener ciertas metas establecidas por las ciencias computacionales en el campo de la inteligencia artificial. Algunas de esas metas son: diseñar nuevas metodologías computacionales para la solución de problemas, la dirección de análisis, diseño e implantación de

[3] Sony Global- AIBO Global Link, 2006.
[4] Coppin. *Artificial Intelligence Illuminated*, 2004, p.p. 25-30.
[5] Kim, et al. "Multi-Agent Systems: A Survey from the Robot-soccer Perspective". p. 3-5.

proyectos de desarrollo tecnológico computacional capaz de generar nuevo conocimiento, entre otras.

La inteligencia artificial se puede ver como el estudio de métodos que se pueden construir por medio de agentes inteligentes. Los agentes inteligentes tienen como propiedades fundamentales: inteligencia, autonomía, habilidad de aprender, cooperación, entre otros. La habilidad para aprender, es la característica de los agentes de interpretar nueva información del ambiente.

El aprendizaje permite a los agentes mejorar su comportamiento al realizar tareas muy específicas. El agente es capaz de aprender de su propia experiencia al ejecutar una acción y de esta manera no cometer los mismos errores en el futuro. La manera en la cual se intenta simular el actuar humano para la generación del aprendizaje por refuerzo es conocido como el aprendizaje básico humano pues genera comportamientos para brindar soluciones a los agentes[6].

Lo anterior representa una gran motivación pues la investigación tiene como objetivo primordial la realización de un trabajo directamente en el motor central de la inteligencia artificial, el aprendizaje colaborativo para agentes en un medio dinámico. Dicho motor representa un gran avance para la etapa humana en el campo de la robótica. Finalmente estos cambios representan beneficios esenciales para la vida del hombre.

[6] Sutton, et al. *Reinforcement Learning. An Introduction.* p. 56-60.

II. ANTECEDENTES Y ESTADO DEL ARTE ACERCA DEL TRABAJO DE INVESTIGACIÓN.

1. INTRODUCCIÓN DE LOS ANTECEDENTES DEL TRABAJO DE INVESTIGACIÓN.

Este capítulo tiene por objetivo llevar al lector al entendimiento del estado del arte en el cual se encuentra tanto en nuestra institución así como las universidades mundiales. Para ello este capítulo se divide en tres partes fundamentalmente: trabajo de investigación en nuestra institución, los trabajos y definiciones de lo que hoy se conoce como aprendizaje por refuerzo y finalmente los trabajos relacionados en este tema que hay actualmente en la liga mundial de *Robocup*[7].

En primera instancia tendremos el trabajo que se ha realizado por más de cinco años en lo que hoy se conoce como el Proyecto de los Robots Cuadrúpedos en el Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Estado de México (*I.T.E.S.M. CEM*)[8]. La idea fundamental es conocer la historia, la visión de investigación y el trabajo futuro que tiene este proyecto. Con lo anterior, se tiene la meta de encontrar un punto en el cual se solucione un problema en específico que presenta este proyecto dadas las herramientas generadas y hechas por el mismo equipo de trabajo que conforma el proyecto.

Dado lo anterior, este capitulo analizará las definiciones básicas de lo que se conoce como aprendizaje por refuerzo. Además éste detallará los conceptos fundamentales en el campo de aprendizaje por refuerzo como lo son: políticas, funciones de recompensa y valor, modelo del ambiente y retroalimentación por evaluación. Lo anterior obedece a que el lector interprete la relación de este tipo de aprendizaje como una posible solución de investigación que necesita el equipo del proyecto de los Robots Cuadrúpedos.

Finalmente, este capítulo describirá la investigación sobresaliente de la liga mundial de *Robocup*. Es importante señalar que como esta liga se encuentra dividida en varias categorías, el lector podrá analizar los trabajos más importantes en cuanto a aprendizaje por refuerzo se refiere por categoría. Actualmente se han desarrollado trabajos en las siguientes ligas:

1. *Robocup* en casa.
2. *Robocup* en rescate.
3. Liga de simulación.
4. Liga de humanoides.

[7] Robocup Official Site, 2006.
[8] Proyecto de los Robots Cuadrúpedos, 2006.

2. TRABAJO DE INVESTIGACIÓN Y AVANCES DEL PROYECTO ROBOTS CUADRÚPEDOS.

A. INICIOS E HISTORIA DEL EQUIPO TECRAMS.

El proyecto Robots Cuadrúpedos conocido como *TecRams*[9] del Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México (ITESM CEM), utiliza los robots *Sony Aibo* modelo *ERS7*[10] para competir en la federación mundial *Robocup*[11] en la liga de *Four Legged*. Los robots son de forma canina y tienen una altitud de 30 cm. por 27 cm. de alto. Éstos cuentan con una cámara de color y un micrófono en su cabeza, además tienen una bocina para emitir sonidos, una tarjeta inalámbrica de comunicación, entre otras características. Para su programación, cada robot es manejado como una computadora independiente la cual tiene su propio *hardware* y *software*.

TecRams participó por vez primera en Robocup 2002 en Fukuoka, Japón, como único equipo ibero americano. Con sólo 3 meses de trabajo, después de ser aceptados en la competencia, TecRams diseño e implementó un equipo de fútbol con los Sony Aibo ERS-210 para la categoría Four Legged. De ahí en adelante, el equipo ha participado en las siguientes competencias:

Competencia 2002:

- World Cup Robocup 2002. Fukuoka, Japón.

Competencias 2003:

- World Cup Robocup 2003. Padova, Italia.
- First American Open 2003. Pittsburgh, Estados Unidos.

Competencias 2004:

- World Cup Robocup 2004. Lisboa, Portugal.
- Latin American Open 2004. Estado de México, México.

Competencias 2005:

- First Mexican Robotics Contest 2005. Morelos, México.
- First Latin American Open 2005. Maranhao, Brasil.

Competencias 2006:

- Second Mexican Robotics Contest 2006. Ciudad de México, México.
- Second Latin American Open 2006. Santiago, Chile.

[9] Proyecto de los Robots Cuadrúpedos, 2006.
[10] Sony Global- AIBO Global Link, 2006.
[11] Robocup Official Site, 2006.

Competencias 2007:

- Third Mexican Robotics Contest 2007. Puebla, México.
- Third Latin American Open 2007. Monterrey. México.
- World Cup Robocup 2007. Atlanta, Estados Unidos.

"El Aibo de Sony® ha sido usado como una plataforma de investigación en los laboratorios de IA alrededor del mundo. Una de las tareas más estudiadas en IA con esta plataforma es el fútbol robótico, un juego competitivo muy parecido al fútbol humano, pero jugado con robots móviles autónomos. El fútbol robótico provee grandes oportunidades a la investigación en IA, desde que este conlleva a una serie de problemas prototípicos para muchas otras aplicaciones de robótica más serias. Competencias anuales de fútbol robótico han atraído gran numero de investigadores en IA y han agregado mucha emoción al campo de la robótica" [9].

Figura 2. Robot Sony AIBO Modelo 210 con el control del esférico en el terreno de fútbol.

En liga de robots cuadrúpedos se utiliza el robot Aibo de Sony® como se muestra en la figura 2. Cabe mencionar que el hecho de utilizar la misma plataforma de hardware, hace a la liga de robots cuadrúpedos, la más competitiva de Robocup porque el ganador se distingue exclusivamente por la innovación en los métodos utilizados para reconocimiento, aprendizaje, planeación, cooperación, etc.

B. TRABAJOS E INVESTIGACION DEL PROYECTO.

La manipulación de cada componente físico del robot se realiza mediante un sistema operativo propietario de *Sony* llamado *AperiOS*. Éste es un sistema operativo que es controlado mediante una serie de funciones que en conjunto así forman una librería. *Sony* ha desarrollado una librería conocida como *OPEN-R*[12] para mandar al *AperiOS*. En otras palabras, el *OPEN-R* sirve para el desarrollo de programas y manipular al robot de manera directa [Figura 3].

[12] [AIBO SDE] official web site, 2006.

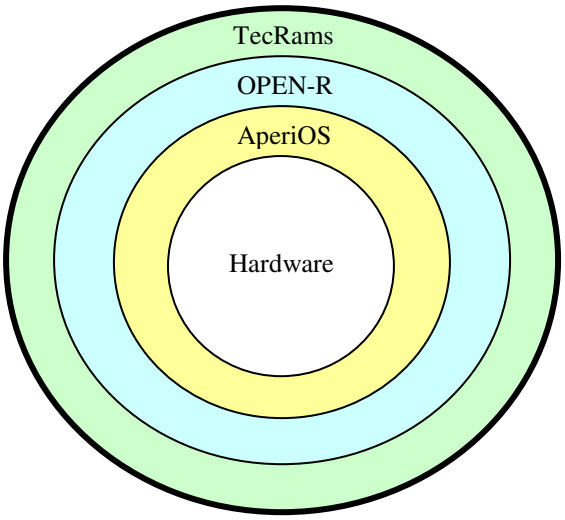


Figura 3. **Arquitectura de código actual de TecRams.**

Para lograr los diferentes objetivos que se tienen sobre los robots, el proyecto de los Robots Cuadrúpedos está dividido en cuatro principales módulos: visión, locomoción, comunicación y decisión:

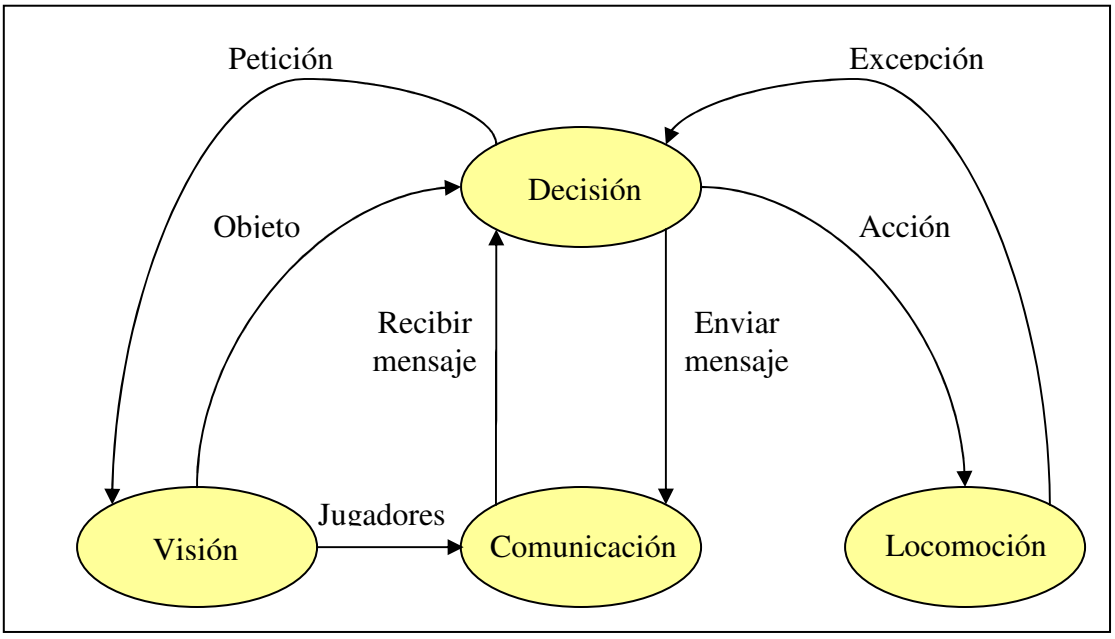


Figura 4. **Organización de módulos en el proyecto TecRams.**

- El área de visión se encarga de la detección de colores en la cancha de acuerdo a la iluminación, además busca reconocer los objetos con el fin de localizar cada uno de ellos y obtener su posición aproximada en el terreno de juego.
- El área de locomoción se encarga de manejar cada sensor y servo motor del robot con el fin de obtener movimientos diversos con las patas y cabeza del robot como son: el mejor caminado posible, patadas, pases y festejos.

- El área de comunicación se encarga de crear diferentes paquetes por medio del uso de *UDP* como de *TCP* con el fin de cambiar algún estado en el que un agente se encuentre; esto es, el robot tiene la capacidad de recibir mensajes de manera inalámbrica con el fin de cambiar algún comportamiento en el momento del juego.
- El área de decisión se encarga de la manipulación de acciones del robot de acuerdo a la información del ambiente por medio de las entradas tanto del módulo de visión como de locomoción para que de esta manera se obtenga dicha acción para el robot de acuerdo a la dinámica del juego [Figura 4].

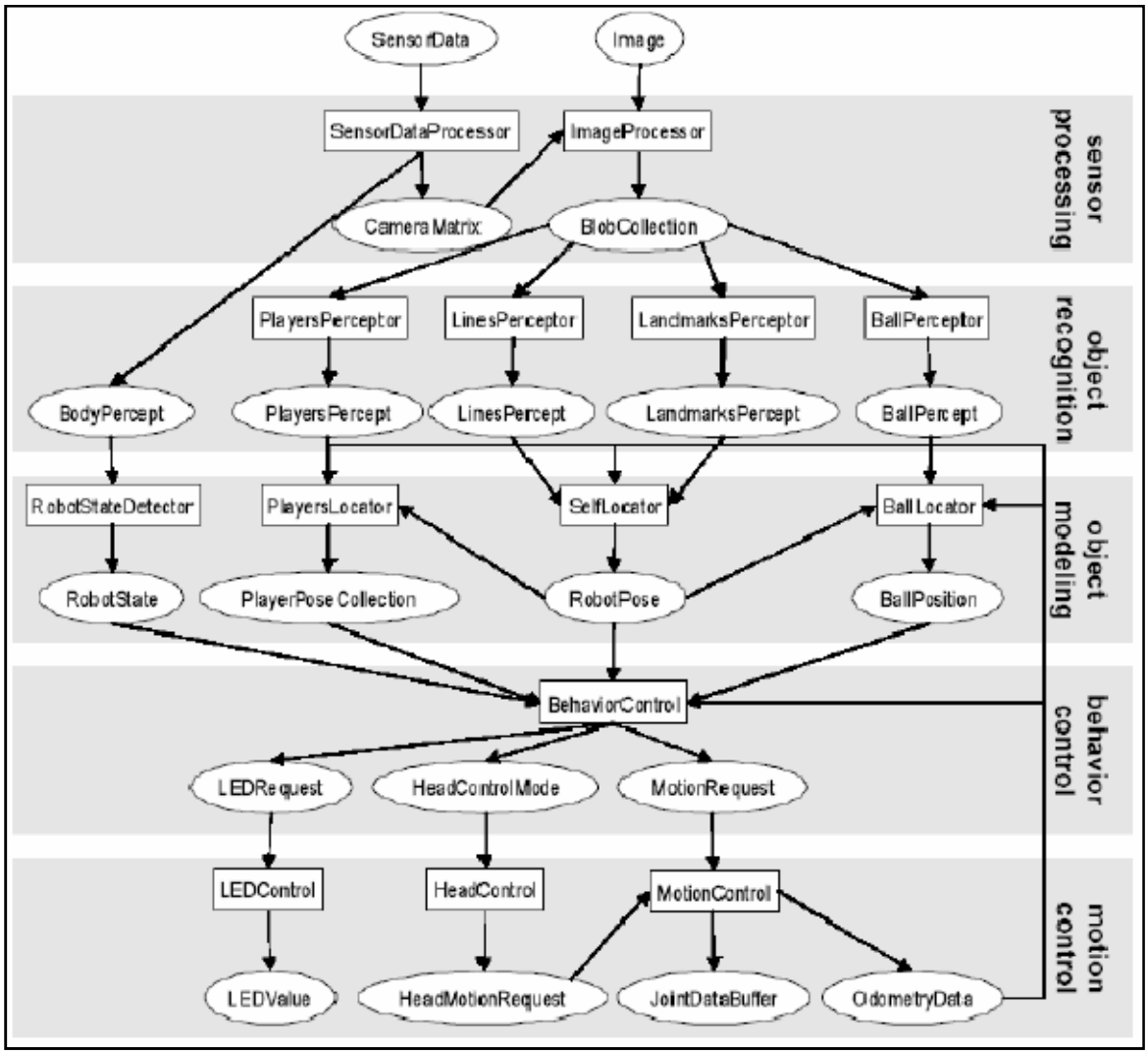


Figura 5. **Organización de módulos en el proyecto GermanTeam.**

Actualmente, el módulo de decisión tiene como objetivo primordial la generación de comportamientos de un sistema multiagente en un medio dinámico para jugar fútbol. Éstos son programados de manera manual, lo que provoca que sean eminentemente reactivos, esto es, a pesar de mantener posición fija, cada agente trata de conseguir el esférico para tirar a gol.

Estos comportamientos están divididos de acuerdo la posición del agente: portero, defensa, medio y delantero. La administración y análisis de los comportamientos suele ser muy complejo, por ejemplo en la figura 5 se puede ver la complejidad de la división de comportamientos del equipo campeón del 2004, el GermanTeam. Para resolver el problema de la complejidad y lograr el objetivo principal, el equipo TecRams cuenta con dos diferentes maneras de comportamientos:

1. Generación de comportamientos por medio de la simulación.
2. Generación de comportamientos por medio de herramientas especiales para la generación de código en el agente físico.

C. RESULTADOS Y TRABAJO FUTURO DEL EQUIPO TECRAMS.

El proyecto de los Robots Cuadrúpedos, como herramientas especiales, utiliza dos fundamentalmente. La primera de ellas es conocida como *XABSL*[13]. El *XABSL* es una herramienta generada por el equipo *GermanTeam*[14] de la liga *Robocup*[15]. Ésta utiliza archivos con formato *xml* para la generación de comportamientos, los cuales son compilados e interpretados con comandos específicos del *OPEN-R*[16]. El equipo *TecRams*[17] usa esta herramienta para modificar los comportamientos básicos de los cuatro agentes principales en el campo de juego.

El *XABSL [18]* es utilizado para que los robots Sony *AIBO* tengan ciertos comportamientos en particular. En general se utiliza para que éstos puedan jugar fútbol. Este programa es una interfaz en XML que le permite al programador usar las capacidades del *AIBO* para generar comportamientos [Figura 5].

Existen reglas que rigen el contexto en el que se desenvuelve el juego que también deben ser tomadas en cuenta en el diseño de los módulos del robot. Dichas reglas son aplicadas por árbitros humanos y son notificadas al robot por medio de un software llamado *GameController*, distribuido por la federación.

Este software permite penalizar a los robots, así como de avisar cuando el inicio, finalización del juego y de cuando se ha cometido una anotación indicando el equipo anotador. Las notificaciones sirven para realizar algún tipo de comportamiento en el robot, como podría ser por ejemplo el festejar cuando un equipo ha anotado un gol.

Algunas de las reglas más usualmente aplicadas en el juego son:

- Los jugadores defensivos nunca pueden entrar a la zona de su propia portería. En caso de que así fuese, son sancionados con 30 segundos fuera del juego.

[13] XABSL: The Extensible Agent Behavior Specification Language. 2006.
[14] GermanTeam : Home, 2006.
[15] Robocup Official Site, 2006.
[16] [AIBO SDE] official web site, 2006.
[17] Proyecto de los Robots Cuadrúpedos, 2006.

- No se pueden empujar los robots, en caso de que se estanque el juego porque están atorados, se remueve solamente a los robots que no estén viendo la pelota, es decir, que no tienen intención de ir por la pelota y solo están estorbando.
- Si el portero se encuentra en su área, no puede ser tocado por un robot que pretenda anotar.

Para que un robot pueda ir a atacar es necesario poder reconocer la pelota, decidir cuáles movimientos debe hacer para llegar a ella, comunicarse en el transcurso de su movimiento con sus compañeros, pero lo más importante es poder definir donde esta parado, porque de ahí parte todo el funcionamiento del juego. Las acciones están de alguna manera ligadas a la posición que juega el robot en el terreno. La figura 6 muestra el campo de juego con agentes físicos listos para iniciar un partido de fútbol.

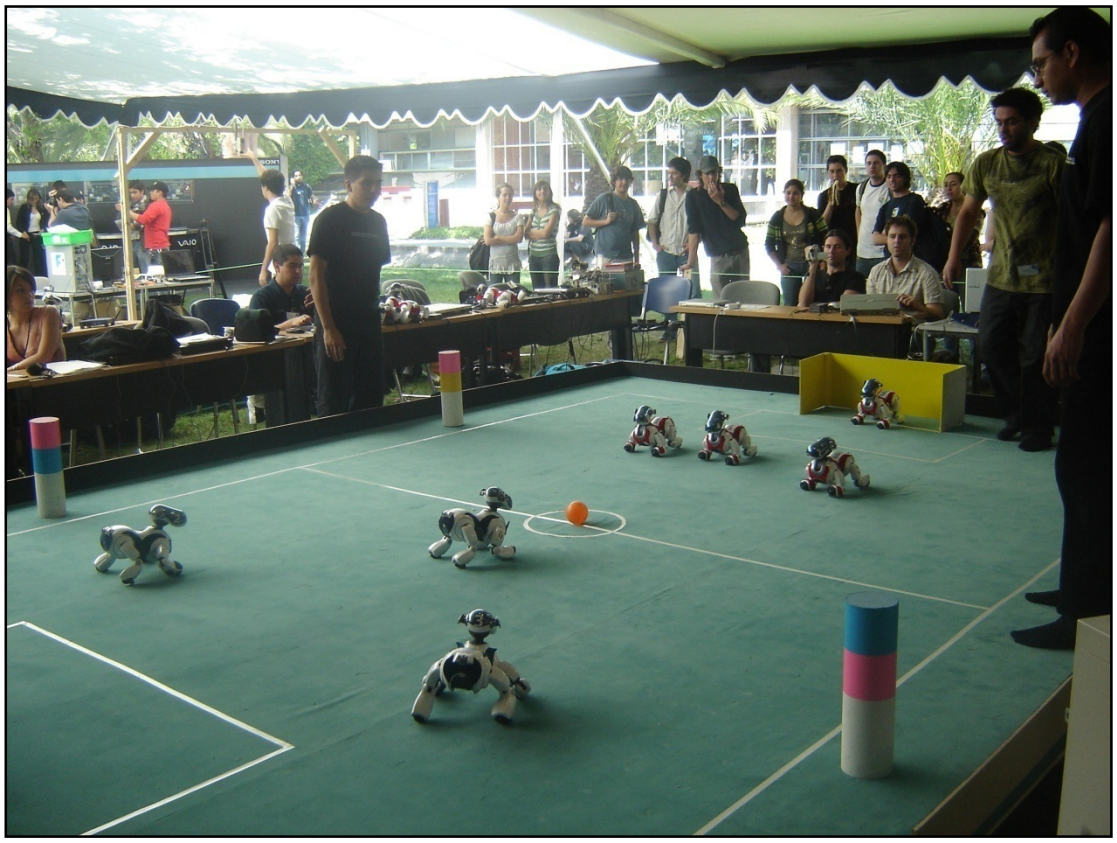


Figura 6. Terreno de juego utilizado por los Sony Aibo para jugar fútbol.

Además, el robot debe desenvolverse en un entorno bien definido pero esto no indica que sea sencillo reconocerlo. El ambiente es altamente dinámico ya que es un juego muy similar al fútbol soccer. El partido es jugado entre dos equipos que constan de cuatro robots; los partidos están compuestos de dos tiempos de diez minutos cada uno, además los robots deben estar uniformados como se muestra en la figura 6, lo cual permite a los robots reconocer a los contrincantes para poder evitar colisiones entre ellos. Al finalizar el primer tiempo los equipos deben alternar sus uniformes con la finalidad de ser ecuánime con los colores que se deben reconocer en orden de detectar al contrincante.

La segunda de ellas se genera por medio de programación genética. *ABG*[18], herramienta creada por el equipo de investigación, TecRams. Ésta crea comportamientos aleatorios de manera genética en archivos con formato de *xml*. Estos archivos son interpretados por la herramienta de simulación *XML Behaviour Control*[19]. De acuerdo al lenguaje específico para los comportamientos, el programa lee cada archivo y evalúa la sintaxis de cada uno de ellos y después valida la información, para obtener objetos de instancia del control interno del sistema [Figura 7].

Figura 7. **Interfaz gráfica del XML Behaviour Control.**

Técnicamente, la programación genética utiliza como cromosoma un conjunto de comportamientos, cada uno contiene una serie de estados, transiciones y acciones complejas. Cada cromosoma en cada generación es evaluado por una función de evaluación. La función de evaluación tiene como propósito final entregar una calificación pertinente y adecuada a cada cromosoma de una generación. De esta manera, el sistema después ordena todos los cromosomas mejor evaluados. Finalmente, éste genera una nueva generación con los mejores cromosomas evaluados y de aquéllos que fueron modificados por mutación [Figura 8].

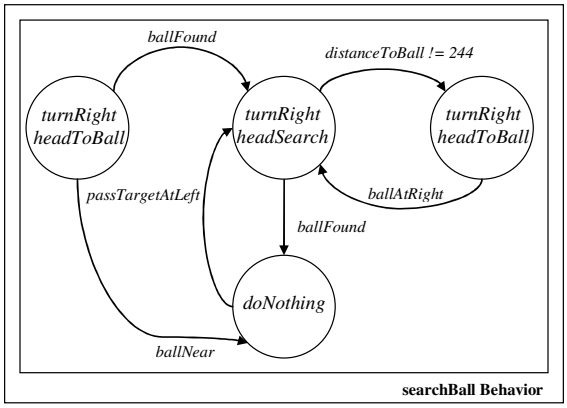


Figura 8. **Cromosoma de una generación representando un comportamiento de buscar la pelota.**

[18] Tapia, et al. "Automatic behavior generation in a multi agent system through evolutionary programming", p. 2-3.
[19] Vega, et al. "Major behavior definition of football agents through XML" , p. 4.

3. DEFINICIONES REFERENTES DEL APRENDIZAJE POR REFUERZO.

Una de las principales áreas de estudio de la inteligencia artificial es el estudio y construcción de maquinas capaces de reconocer patrones de forma automática para discernir acciones. Seria muy practico contar con un equipo capaz de reconocer de forma precisa patrones como el habla, las huellas digitales, secuencias de ADN, entre otras. Al estudiar la complejidad inherente a los problemas de reconocimiento, ganamos mayor conocimiento y apreciación de los sistemas de reconocimiento de patrones del ser humano. Para algunos problemas, tales como: reconocimiento visual y del habla, nuestros diseños deben verse influenciados por el conocimiento de cómo éstos se desempeñan en la naturaleza, tanto en los algoritmos que implementamos como en el diseño de hardware de los dispositivos que intervienen en la aplicación.

Hoy en día, la habilidad de las máquinas para percibir su entorno es muy limitada, a pesar de que existen una variedad en codificadores de luz, sonido, temperatura, etc. en señales eléctricas. Cuando el ambiente es cuidadosamente controlado y las señales tienen una interpretación simple, como en el caso de los dispositivos de entrada de las computadoras, los problemas de percepción se hacen triviales, pero cuando trabajamos con entradas de datos que no están controladas entramos en problemas de interpretación de datos, por ejemplo: para una computadora es sencillo leer un disco óptico con información estructurada, pero es muy complicado reconocer caracteres escritos a mano. De acuerdo a lo anterior, los sistemas de aprendizaje fueron diseñados para generar comportamientos que dependan de la evolución del agente sin importar el medio en el cual está inscrito. Los sistemas de aprendizaje pueden ser agrupados en diferentes formas generales:

1. *Aprendizaje supervisado*: El aprendizaje supervisado está basado en lo que se llaman las fronteras de decisión. Para poder establecer las fronteras de decisión es necesario trabajar con patrones conocidos en una fase conocida como aprendizaje supervisado, ya que se provee de una clase a la que pertenece el patrón dado, es decir, un experto asignar una etiqueta o costo de una categoría a cada patrón del conjunto de patrones de entrenamiento y se busca reducir la suma de los costos para estos patrones. Con ello, se genera una frontera decisión como base para la toma de decisiones del agente, por lo que el agente antes de aprender, conoce todas y cada una de estas fronteras y de esta manera durante su proceso de aprendizaje recorre aquéllas que fueron supervisadas anteriormente.

2. *Aprendizaje no supervisado o clustering*: En esta modalidad no contamos con ningún conocimiento del experto, por lo que no contamos con información acerca de las clases de los objetos. Los patrones de entrenamiento forman *clusters* o “agrupamientos naturales” que por lo general son definidos de forma hipotética por el usuario.

3. *Aprendizaje por refuerzo*: La forma más común de entrenar a un clasificador es presentar la entrada y calcular la asignación tentativa para ajustar el clasificador. La única retroalimentación del experto será si esta bien o mal con lo que podremos recompensar o castigar la decisión que el clasificador dio hasta que este siempre de soluciones razonables. De forma análoga, los seres humanos aprenden a hacer las cosas bien a base de estímulos positivos o negativos.

El aprendizaje por refuerzo nació en 1979 gracias a Harry Klopf, quien inició sus investigaciones con base en la teoría heterostática. Esta teoría se genera por medio de sistemas adaptables los cuales utilizan algoritmos básicos como el Q-Learning[20].

El aprendizaje por refuerzo siempre está en busca de los mejores parámetros para una acción [Figura 9]. Para obtener el objetivo anterior, este aprendizaje utiliza un conjunto de comportamientos, los cuales tiene que modificar a través del tiempo. La idea principal del aprendizaje por refuerzo es capturar los aspectos más importantes para un problema de un agente que necesita una meta en un ambiente dinámico. Existen tres tipos de algoritmos de aprendizaje por refuerzo: Programación dinámica, métodos Monte Carlo y métodos de Diferencia Temporal.

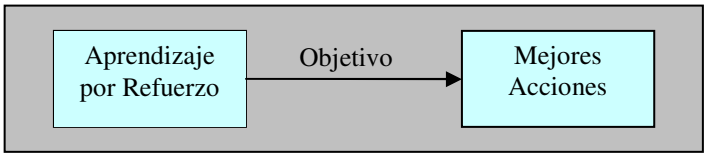


Figura 9. El objetivo principal del aprendizaje por refuerzo.

En términos generales, el aprendizaje por refuerzo es aprender qué hacer, o sea, encontrar las similitudes entre acciones para maximizar las recompensas durante el lapso que dure dicho aprendizaje. Es importante recalcar dos aspectos fundamentales, el primero se debe a que este tipo de aprendizaje obtiene las acciones por medio de algoritmos de programación, por lo que prueba cada una de estas acciones para encontrar la recompensa inmediata justa de la ejecución de ésta en un ambiente determinado y segundo se debe a que las metas que tiene el agente pueden cambiar, esto es, los estados finales en los comportamientos se pueden cambiar pues el ambiente dinámico genera nuevas posturas en el agente y torna cambios internos para su correcto desempeño.

La recompensa que obtiene cada agente puede ser una de dos maneras: búsqueda de prueba y error y recompensa retardada. La búsqueda de prueba y error es conocida como un algoritmo de explotación, pues la acción que es recompensada como la mejor en el tiempo actual se obtiene para cambiar los parámetros de un estado del comportamiento a ejecutar.

La recompensa retardada es conocida como un algoritmo de exploración, pues de manera aleatoria se elige una acción independientemente de la recompensa que tenga y esa se coloca en el agente en su comportamiento a ejecutar. En otras palabras, cuando un agente explota, utiliza su conocimiento actual para obtener su recompensa respectiva y por otra parte, cuando decide explorar, el agente hace una selección de otras acciones posibles que no sean las mejores para llegar a tener una mejor recompensa no inmediata, sino una que ayude para encontrar dicha acción en el futuro.

Los elementos principales del aprendizaje por refuerzo son:

1. *Políticas:* es el mapeo de los estados actuales con sus acciones respectivas.
2. *Función de recompensa:* otorga un número inalterable por estado de un agente.
3. *Función de valor:* es la función que indica si una acción es correcta para el futuro por medio de un número que puede cambiar con el tiempo.

[20] Sutton, et al. *Reinforcement Learning. An Introduction.* p. 56-60.

4. *Modelo del ambiente:* son modelos probabilísticos usados para la planeación.
5. *Retroalimentación por evaluación:* es la evaluación de las acciones realizadas por el agente después de que han sido entrenadas todas y cada de las acciones hechas anteriormente.

Finalmente, el aprendizaje por refuerzo busca la relación de acciones con estados con el valor más alto en el futuro y no con el la recompensa más alta, pues de esta manera se logra que las acciones que aprende un agente a través del tiempo sean las mejores de acuerdo a las metas establecidas en el medio en que actúa. Lo anterior describe la manera simulada en que vive un humano, pues gracias a la prueba y error, las personas somos capaces de aprender y entender las consecuencias de la acción realizada. Así se obtiene un aspecto primordial de la inteligencia artificial, el aprendizaje por refuerzo usado en agentes para conseguir objetivos definidos de un ambiente.

A. POLITICAS.

Una política define la manera en cómo un agente aprende su comportamiento durante un tiempo determinado. La política utilizada en aprendizaje por refuerzo se define como el mapeo de estados percibidos del ambiente por el agente hacia acciones a realizarse por dichos estados. En términos del estudio de la Psicología, una política es una regla de estímulo – respuesta o una asociación.

Generalmente se define la política como una función o una tabla de búsqueda. Es importante señalar que se considera como un eficiente banco de políticas cuando hay procesos de búsqueda específicos para encontrar la adecuada política en cada caso que suceda con el agente y su relación con el ambiente. Finalmente es considerado que las políticas representan la columna o base fundamental del aprendizaje por refuerzo, pues son los que determinan el comportamiento final de un agente.

B. FUNCIÓN DE RECOMPENSA.

Una función de recompensa define la meta y objetivo del problema del aprendizaje por refuerzo. Esta función tiene por objetivo regresar un número simple de manera inmediata dentro del mapeo de los estados percibidos y la acción realizada. En este punto es importante denotar que el objetivo de cualquier agente dentro de un ambiente es maximizar la recompensa total que recibe en largo plazo, esto es, cada vez que el agente realiza una acción recibe la recompensa de esta función y al tratar de maximizarla en el largo plazo llegará eventualmente a cumplir el objetivo deseado.

La función de recompensa define como correcto o incorrectos los eventos que realiza el agente. En otros términos, la recompensa describe el placer o el sufrimiento que recibe el agente dada una acción ejecutada.

Otro aspecto importante de lo anterior es que esta recompensa no puede ser modificada por ningún motivo por el agente, pues generalmente se obtiene después de aplicar la política exacta que merece el agente.

En la Figura 10 puede observarse la recompensa inmediata para una regla que se puede disparar en algún momento dado del aprendizaje. En esta figura se demuestra en la etiqueta *reward* en donde se recompensa que si esta regla se dispara, el agente es castigado o recompensado por ello.

```
<ql-rule id="2" name="OVERTIME and position greater and ball found" useTimer="18">
    <and>
        <ql-event name="agent" value="positionX" function="greaterThan(0)"/>
        <ql-event name="eventBF" value="VISION-ballFound" function="isActive()"/>
    </and>
    <reward goWell="true" goBad="false"/>
    <debug text="Position = " info="positionX" />
    <debug text="Time is Over: reward position + ballFound"/>
</ql-rule>
```

Figura 10. **Recompensa por política o regla en el sistema BADRL.**

C. FUNCIÓN DE VALOR.

La función de valor indica que tan correcta o incorrecta es una acción pero en el largo plazo, esto es, cuál es la cantidad acumulada de la acción ejecutada por el agente en el futuro. Para diferenciar esta función con la anterior, cabe recordar que la recompensa es un valor inmediato e intrínseco, y el valor indica el grado de deseabilidad en el futuro de todos los estados sobre las recompensas recibidas.

Un ejemplo de lo anterior se puede denotar de la siguiente manera: si un agente *A* recibe una recompensa *R* gracias a la política *P*, entonces el Valor *V* cambia por cada *R* que va recibiendo A mientras realiza el proceso de aprendizaje.

D. MODELO DEL AMBIENTE.

El ambiente es definido como el campo o lugar en el cual se encuentra inscrito un agente. Para que exista un modelo es necesario que el agente pueda tomar decisiones y actuar sobre su medio ambiente por medio de un modelo.

Además de lo anterior se obliga a que el agente tenga la capacidad de percibir los cambios que se efectuaron sobre un modelo que es capaz de modificar su estructura de acuerdo a las acciones de dicho agente. Cada vez que existan nuevos cambios sobre ésta, los agentes deben percibir esos cambios de acuerdo a los eventos que realiza el agente en el ambiente.

E. RETROALIMENTACIÓN POR EVALUACIÓN.

La retroalimentación por evaluación es una función representativa del aprendizaje por refuerzo. Para que exista aprendizaje por refuerzo, obligatoriamente el sistema debe entrenar la información del medio para evaluar acciones.

Lo anterior implica necesariamente que el sistema no debe instruirlas, esto es, el sistema debe evaluar todas las acciones generadas y obtener aquéllas consideradas como ideales. Se le llama acción ideal, a todas aquéllas que cumplen con una política en cada corrida del algoritmo de aprendizaje.

Por una parte la retroalimentación por evaluación indica qué tan buena es la acción tomada en el momento por un agente dentro del ambiente de un sistema, sin embargo no indica si es la mejor o la peor. Esta evaluación es una base de métodos evolutivos para funciones de optimización. Por otra parte la retroalimentación por instrucción indica si una acción es correcta independientemente si es tomada o no. Esta retroalimentación es la base de lo que se conoce como el aprendizaje supervisado usado generalmente para reconocimiento de patrones, redes neuronales, identificación de sistemas, entre otros.

4. TRABAJO SOBRE APRENDIZAJE POR REFUERZO EN LA ORGANIZACIÓN *ROBOCUP*.

Desde 1997, año en que formalmente aparecieron por vez primera una serie de competencias y conferencias por parte de la liga de *Robocup*[21], las investigaciones sobre la generación de inteligencia artificial han generado varias soluciones para completar dicha tarea. Una de las soluciones, que más han sido desarrolladas desde principios de la década pasada, es el aprendizaje por refuerzo.

La competencia mundial de robots (*Robocup*) se ha establecido exitosamente como una plataforma estándar para la investigación de inteligencia artificial y robótica. Históricamente este tipo de problemas estándar han dado origen a grandes avances tecnológicos y ahora al utilizar el juego de fútbol como problema común, se desarrollan y ponen a prueba principios de agentes autónomos, sistemas multiagente, colaboración, competencia, desarrollo de estrategias, razonamiento en tiempo real e inclusive desarrollo de hardware[22].

Particularmente el juego de fútbol cubre con varias áreas dentro de la inteligencia artificial e ingeniería como son adquisición de datos en tiempo real, comportamientos reactivos, definición de estrategias, aprendizaje, planeación en tiempo real, sistemas multiagentes, reconocimiento de patrones, visión, toma estratégica de decisiones, control, inteligencia entre muchas otras.

El juego del fútbol soccer puede ser visto como un sistema mutiagente, altamente competitivo y a la vez cooperativo. Además, los jugadores deben realizar diferentes comportamientos dependiendo de la situación en la que se encuentren. Esta situación es sumamente dinámica

[21] Robocup Official Site, 2006.
[22] Vega, et al. “Major behavior definition of football agents through XML” , p. 4.

debido a las características del mismo juego: la posición, posesión y control de la pelota, el compartimento de un agente específico y el comportamiento de los demás jugadores pues todos y cada uno están en constante movimiento.

El aprendizaje por refuerzo se puede ver con un algoritmo capaz de entregar a través del tiempo el mejor resultado posible para un objetivo específico. Debido a la gran gama de problemas que pueden ser resueltos por este tipo de algoritmos, los investigadores que se encuentran dentro de la liga de *Robocup* lo han utilizado en varios aspectos de la robótica actual. De acuerdo a las diferentes ligas, que se encuentran actualmente en *Robocup*, la liga de simulación es considerada la más avanzada, pues se limita a la generación de comportamientos y estrategias para lograr vencer a los otros equipos con los que se está compitiendo. Dentro de esta liga, se han generado varias investigaciones que utilizan el aprendizaje por refuerzo.

A. TRABAJOS TEÓRICOS SOBRE APRENDIZAJE POR REFUERZO.

Desde hace una década, la investigación en inteligencia artificial ha tenido especial interés en el uso de algoritmos para robots cooperativos o sistemas multiagentes. De hecho, existen diferentes artículos de investigación sobre este tema en específico. Uno de los más importantes es el desarrollado por Jong-Hwan Kim y Prahlad Vadakkepat en su artículo Multiagent Systems: A Survey from the Robot-soccer Perspective[23].

Este trabajo de Kim revisa de manera profunda aquellos sistemas que pueden ser utilizados para agentes que juegan fútbol. En especial se dedica a revisar este deporte pues nos dice que el estudio sobre comportamientos con un desempeño similar al de los humanos para la inteligencia artificial sobre agentes autónomos representa un gran reto e interés para los investigadores actualmente. Ahora bien, ya que se tiene el tema a analizar, Kim encontró que los sistemas multiagente pueden controlar la totalidad de los agentes de una manera completa y correcta. Por lo que a partir de aquí se generaron diferentes tipos de investigaciones sobre el desarrollo de algoritmos para agentes que juegan fútbol. En términos generales, la contribución de Kim fue reconocer que un sistema multiagente puede controlar agentes cuyo objetivo es jugar fútbol.

El objetivo primordial del fútbol es anotar la máxima cantidad de goles en la portería contraria, razón por la cual Veloso y Stone utilizan de nueva cuenta el aprendizaje por refuerzo para lograr este objetivo[24]. Un equipo simulado logra anotar la máxima cantidad de goles en el menor tiempo posible. Para ello, el equipo utiliza de una gama de acciones para que en conjunto logren dicha meta. Sin embargo, en ese momento no existía un equipo con cual compararse por lo que fue necesario utilizar un equipo que jugara de manera aleatoria el cual usara dichas acciones anteriores. El resultado que arrojó esta investigación es que un equipo que se maneja por aprendizaje por refuerzo logra anotar más goles que uno aleatorio [Figura 11].

[23] Kim, et al. "Multi-Agent Systems: A Survey from the Robot-soccer Perspective". p. 3-5.
[24] Stone, et al. "Team-Partitioned, Opaque-Transition Reinforcement Learning". p. 2-3.

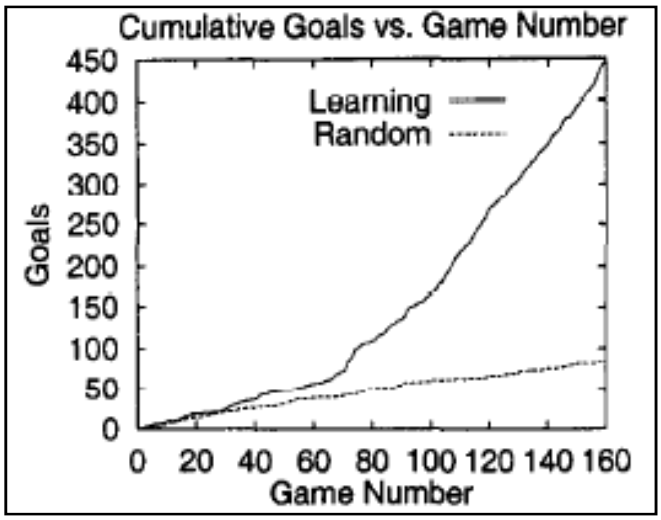


Figura 11. Comparativo un equipo que utiliza aprendizaje por refuerzo frente a uno aleatorio.

Por otro lado, existen investigaciones que van más allá de tener sólo un algoritmo de aprendizaje por refuerzo por agente, esto es, hay trabajos que analizan y generan nuevos datos con nuevas formas de aprendizaje por refuerzo colaborativo en estrategias por equipo. Ejemplo de ello, se puede encontrar en los trabajos realizados por: Salustowicz, Wiering y Schmidhuber en 1998.

Uno de los trabajos más importantes de Salustowicz es el de Learning Team Strategies: Soccer Case Studies[25]. En esta investigación se comparan diferentes algoritmos de aprendizaje para el estudio del aprendizaje en un sistema multiagentes. Este sistema estaba compuesto por dos equipos, y cada equipo estaba compuesto de jugadores de fútbol [Figura 12]. Cada equipo compartía el conjunto de acciones y políticas entre ellos, sin embargo las entradas y los actuadores de cada agente percibían diferentes entradas del ambiente.

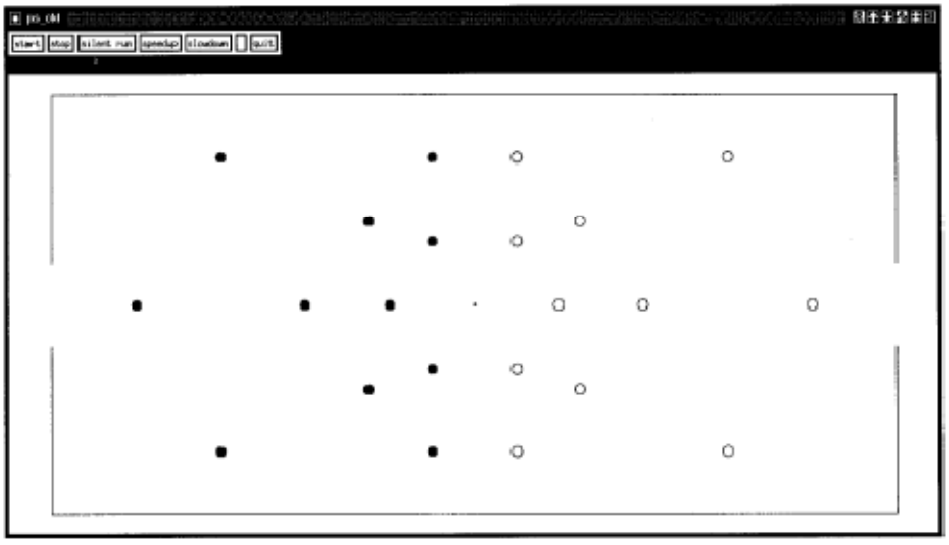


Figura 12. Equipos compuestos por 11 jugadores en el campo de simulación.

[25] Salustowicz, et al. "Learning Team Strategies: Soccer Case Studies". pp. 263-282.

Los resultados finales de esta investigación arrojaron que los algoritmos de diferencia temporal como Q-Learning tienen dificultades para tener un aprendizaje apropiado para compartir información entre varios agentes, sin embargo los algoritmos conocidos como Probabilistic Incremental Program Evolution (PIPE) y Co-Probabilistic Incremental Program Evolution (COPIPE) si encuentran políticas correctas en un tiempo rápido y confiable. Esto debido a que estos últimos algoritmos no dependen de funciones de evaluación de aprendizaje como los de diferencia temporal, como se muestra en la figura 13.

Team size		*GO*	PIPE	CO-PIPE	TD-Q
	Maximum score difference	417	310	192	42
1	Average goals ± st.d.	417 ± 6	320 ± 42	212 ± 97	52 ± 14
	Average *BRO* goals ± st.d.	0 ± 0	10 ± 7	20 ± 10	10 ± 3
	Achieved after games	n.a.	3300	3000	1700
	Maximum score difference	481	359	310	70
3	Average goals ± st.d.	481 ± 8	373 ± 86	324 ± 62	102 ± 14
	Average *BRO* goals ± st.d.	0 ± 1	14 ± 6	14 ± 11	32 ± 8
	Achieved after games	n.a.	3300	3200	1700
	Maximum score difference	364	481	357	154
11	Average goals ± st.d.	367 ± 18	512 ± 129	393 ± 53	212 ± 84
	Average *BRO* goals ± st.d.	3 ± 1	31 ± 23	36 ± 27	58 ± 23
	Achieved after games	n.a.	3100	1900	2500

Figura 13. Resultados comparativos entre PIPE, COPIPE y TD-Q para el comportamiento de pase.

Existen otras investigaciones a cerca del aprendizaje por refuerzo para agentes físicos que juegan fútbol. Una de éstas se ve reflejada en la arquitectura Clay de Tucker Balch[26]. Clay es una arquitectura con algoritmos de evolución para agentes o robots autónomos tienen motores integrados. Es muy importante señalar que el tema de los motores es primordial en esta investigación, pues las acciones que toman son mandadas por un coordinador que tiene el control del aprendizaje por refuerzo por medio de Q-Learning.

El proceso fundamental de esta investigación es que se obtiene una instancia de un objeto llamado *CoordinateLearner* el cual contiene la información de los motores. Esta instancia es usada para crear una instancia de otra clase llamada *CoordinateLearner* quien es el encargado finalmente de usar *Q-Learning*. Con esta información el último coordinador manda las señales a los motores para que sepan qué acción realizar [Figura 14]. Esta investigación es importante ya

[26] Balch, "Integrating RL and Behavior-based Control for Soccer". p. 3.

que brinda una manera de utilizar *Q-Learning* en robots físicos para generar acciones y ciertos comportamientos específicos para un agente.

```
learner
   = new LearnerQ(states, actions, alpha, gamma,
                          randomrate, randomdecay);

reward = new RewardOnScore(m);
assemblages[0]    = move_to_ball;
assemblages[1]    = get_behind_ball;
assemblages[2]    = move_to_backfield;

top_level
   = new CoordinateLearner(assemblages, behind_ball,
                                        reward, learner);
```

Figura 14. Instancias principales utilizadas en la arquitectura Clay.

B. INVESTIGACIONES SOBRE COMPORTAMIENTOS POR MEDIO DEL USO DE APRENDIZAJE POR REFUERZO EN LA LIGA DE SIMULACIÓN DE *ROBOCUP*.

Uno de los primeros trabajos para la liga de simulación, que utilizan este aprendizaje, fue realizado por Matsubara, Noda e Hiraki[27]. El objetivo primordial de este trabajo es la selección de la mejor acción en un momento dado del juego. Las dos acciones que se analizaron fueron el pase y tiro. La función de recompensa, que utiliza este trabajo, obtiene los valores por medio de funciones trigonométricas. Esto es, si el ángulo entre el portero y el delantero que tiene la bola es mayor a noventa grados por lo tanto tiene una recompensa inmediata mayor [Figura 15].

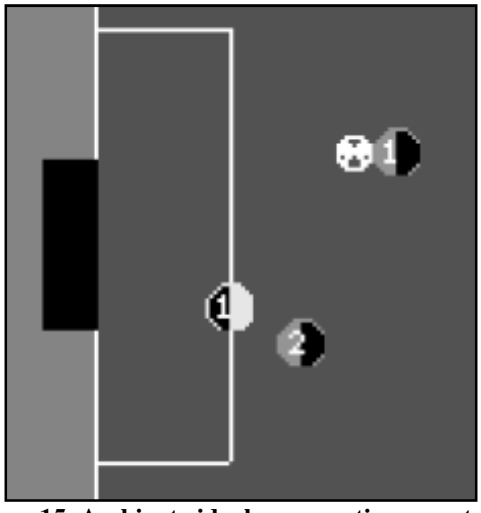


Figura 15. Ambiente ideal para un tiro a portería.

[27] Matsubara, Noda e Hiraki

El aprendizaje por refuerzo es un algoritmo que se ha utilizado para resolver varios problemas. Peter Stone[28] solucionó uno de ellos por medio de los principios fundamentales de este aprendizaje para generar comportamientos específicos sobre jugadores simulados. Cuando la liga de simulación tenía no más de dos años, Stone llegó a la conclusión de que el aprendizaje por refuerzo puede utilizarse también no sólo para acciones simples como pase o tiro, sino también para tomar decisiones en cuanto a conducción se refiere.

Esta investigación es trascendental en el tema del aprendizaje puesto que aquí se manejan dos ramas fundamentalmente: diferentes problemas para ser resueltos por el aprendizaje por refuerzo en un ambiente de jugadores de fútbol y por otro lado, la comparación de los dos métodos principales que utiliza éste: el método de Q-Learning y el de Sarsa[29]. Por una parte el problema planteado radica esencialmente en qué punto un jugador simulado puede conducir la pelota sin que un atacante o adversario se la quite o hasta cuando es capaz de mantener el balón y pasarlo sin que un atacante lo intercepte [Figura 16].

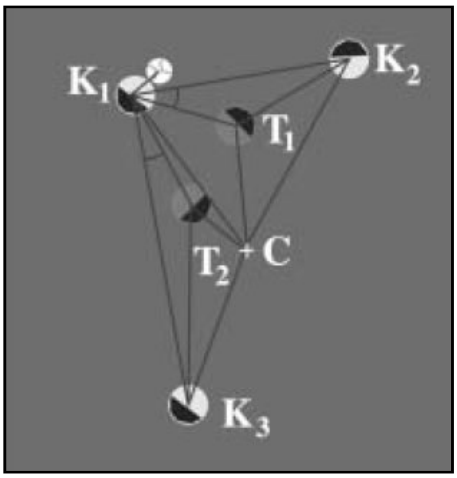


Figura 16. Agente K1 conservando el esférico sin que se intercepte por los contrarios.

Por otro lado, nos permite concluir que para algunos casos, el método de Sarsa resulta más eficiente que Q-Learning, pues las acciones se aprenden más rápidamente. En el caso anterior de la conducción, Sarsa genera en menor tiempo dichas acciones [Figura 17].

Otro de los trabajos que usan el aprendizaje por refuerzo en el ambiente de simulación de Robocup, es el creado por Hatice Köse en su artículo: *Q-Learning based Market-Driven Multi-Agent Collaboration in Robot Soccer*. En esta investigación, se señala la forma en cómo se pueden integrar métodos de manejo de mercado para la correcta colaboración de sistemas multiagente. Este trabajo señala que los algoritmos de diferencia temporal de aprendizaje por refuerzo generan una recompensa de acuerdo a una acción tomada con anterioridad y viendo las consecuencias que ésta generó en el ambiente para un agente en específico. Köse nos indica que para un medio multiagente es necesario definir roles que manejaran los agentes y por medio de métodos de mercadeo, generar las recompensas para cada uno de los agentes de manera autónoma.

[28] Stone et al. "Reinforcement Learning for *Robocup* Soccer Keepaway". p. 4-5.
[29] Sutton, et al. *Reinforcement Learning. An Introduction.* p. 56-60.

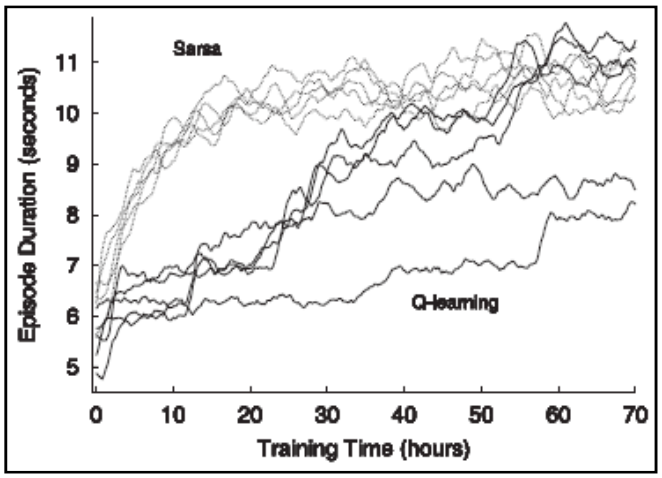


Figura 17. Comparativo de aprendizaje entre Sarsa y Q-Learning.

Un ejemplo de lo anterior, se define para los agentes de fútbol soccer de acuerdo a los roles que pueden manejar en un juego completo. Él define tres roles principales: estar cerca de la bola, ser el más cercano a la portería contraria y tirar. Estos roles fueron definidos ya que el objetivo principal en el juego es la anotación de la bola a la portería. En la figura 18 se puede observar los diferentes roles que manejan los agentes.

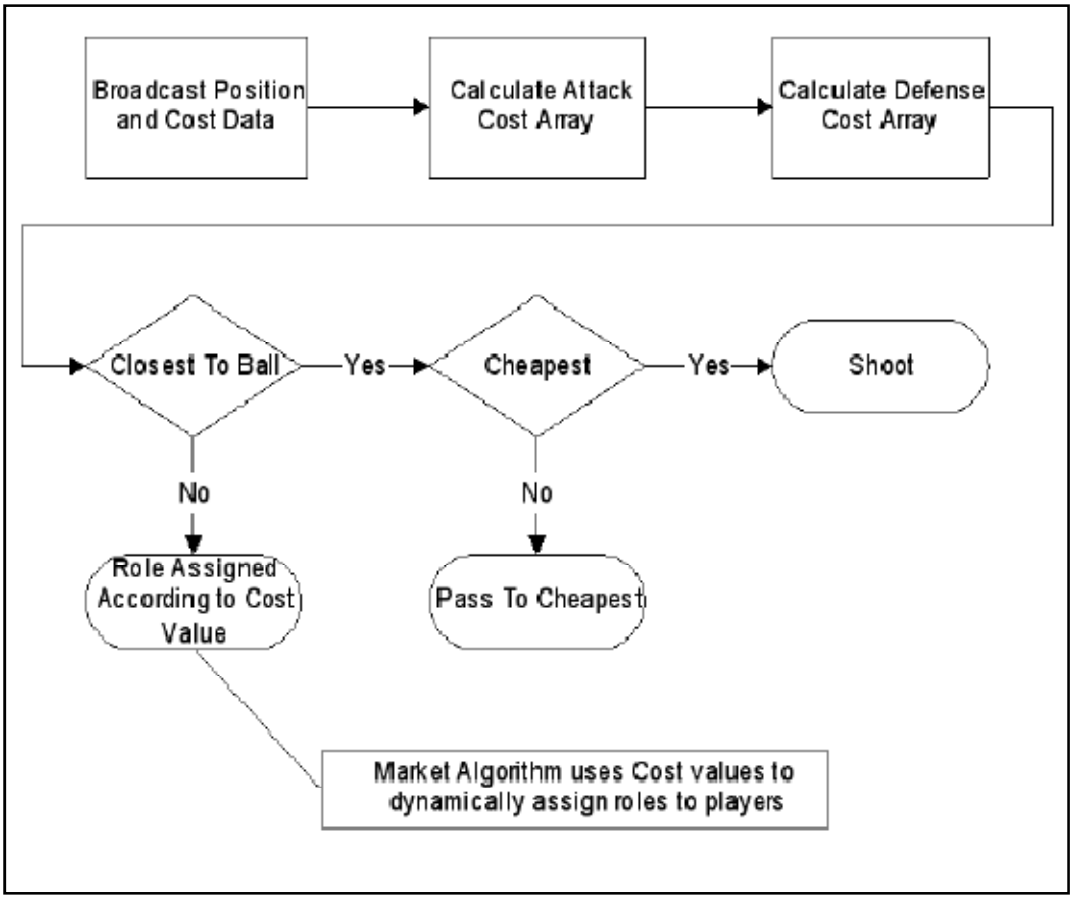


Figura 18. Flujo de tareas para representar tres diferentes roles en un juego de fútbol.

Ahora bien, en el contexto del uso de roles para agentes que juegan fútbol, la investigación concluyó que esta idea funcionó de manera adecuada, ya que durante las pruebas realizadas en el simulador oficial de Robocup Simulator, se observó que se daba recompensa si el agente al tomar un rol específico generaba la respuesta correcta a ese rol, de manera tal que los agentes llegaron a acercase a la bola, dar pase y disparar sólo con el hecho de manejar recompensas por roles. En la figura 19 se podrá observar estas triangulaciones generadas por esta investigación.

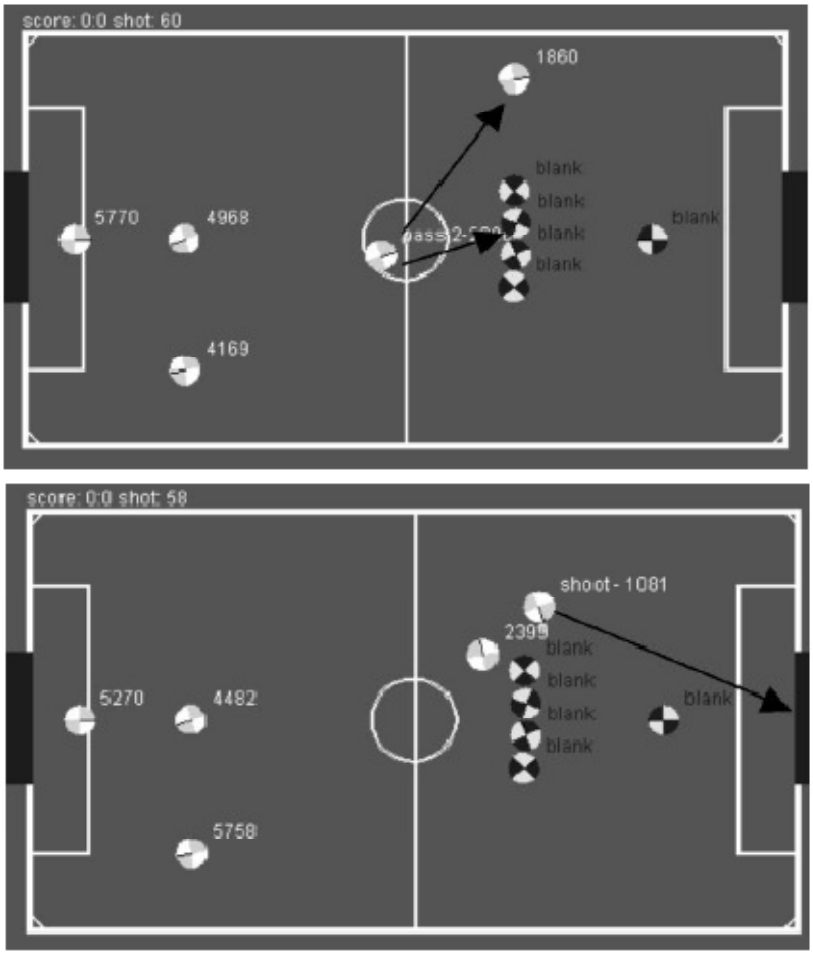


Figura 19. Agentes simulados que realizan pases y tiros de acuerdo a roles programados.

Actualmente se utiliza este aprendizaje en robots físicos como los robots humanoides o los generados para la nueva liga *RoboCup Rescue*[30]. En el campo de los robots físicos de dos extremidades conocidos como humanoides, se ha utilizado los algoritmos de aprendizaje por refuerzo para encontrar la mejor manera de mover los sensores de un robot para un eficiente caminado. Este caminado utiliza la información del exterior para encontrar una odometría exacta del movimiento realizado, como por ejemplo: banderas, líneas, porterías y pelotas

[30] RoboCup-Rescue Official Web Page, 2006.

Lo anterior se ve reflejado en el trabajo de Ogino quien nos muestra que el control de caminado de un robot humanoide es capaz de mantener el equilibrio y reconocer la posición que se encuentra dentro de la cancha, por medio de aprendizaje por refuerzo, que utiliza como acciones, movimientos de los motores y de acuerdo con los objetos que reconoce mediante la cámara, evalúa las acciones por ser aprendidas[31] [Figura 20].

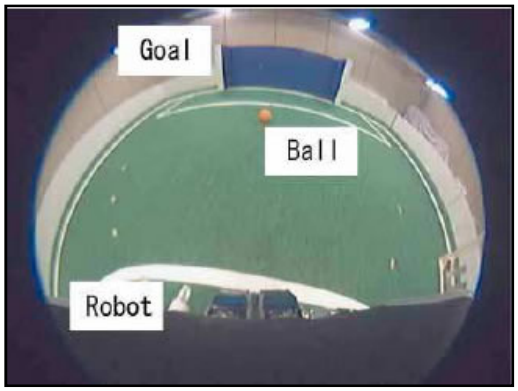


Figura 20. Ángulo de visión de un *humanoide* para obtener la recompensa de sus acciones.

[31] Ogino, et al. "Reinforcement Learning of Humanoid Rhythmic Walikng Parameters based on visual Information". p. 2.

III. PLANTEAMIENTO DEL PROBLEMA Y SU JUSTIFICACIÓN.

1. INTRODUCCIÓN AL TRABAJO Y HERRAMIENTAS UTILIZADAS POR TECRAMS.

Por más de dos años, TecRams, el equipo representativo de robótica del ITESM CEM ha tenido como base el código del *GermanTeam*[32]. El código ha sido modificado y agregado con nuevas herramientas propias del equipo institucional. En el área de decisión, el *XML Behaviour Control*[33] es la aplicación más utilizada para generar comportamientos.

Los comportamientos son creados por medios de archivos *XML* en un formato especial, los cuales pueden ser analizados por medio de la interfaz gráfica de simulación de la herramienta. El análisis se realiza mediante dos ventanas principalmente. En la primera de ellas, se encuentran todas las acciones posibles que puede realizar el agente durante el juego. Cada acción trae consigo un semáforo, el cual, si en algún momento dado del juego ocurre, esa acción prende su semáforo. La segunda de ellas tiene el arreglo de comportamientos que un agente utiliza cuando está jugando. Esta ventana permite visualizar el estado (acciones, transiciones y eventos) en el cual se encuentre en ese preciso momento el agente. Con lo anterior se puede evaluar si un comportamiento o un estado se están mandando a llamar de manera adecuada.

Figura 21. Control de estados por medio del XML Behaviour Control

El *XML Behaviour Control* es un simulador que permite analizar una gran cantidad de variables manejadas por el ambiente como son: opciones de juego, equipo y los robots. Dentro de las

[32] GermanTeam : Home, 2006.
[33] Vega, et al. “Major behavior definition of football agents through XML” , p. 1-2.

opciones de juegos, nos permite visualizar las acciones del mundo que están aconteciendo en ese momento por medio de semáforos. De esta manera, se puede obtener un consenso de acciones y determinar algún comportamiento en especial con dichas acciones para un agente específico. En el momento en que se generan los estados, éstos pueden ser observados por medio de una utilidad que señala en cuál estado se encuentra un robot. En la figura 21 se puede observar la interfaz gráfica para controlar los estados de cada agente en el simulador *XML Behaviour Control*.

El *XML Behaviour Control* se puede ver como una plataforma de definición de comportamientos de robots jugadores de fútbol, separada de la plataforma de programación nativa de los robots. Ésta permite generar y probar estrategias de juego rápidamente. El sistema es utilizado por estudiantes de primeros semestres de carreras de computación sin experiencia en robótica o computación y son capaces de mejorar comportamientos existentes, comprendiendo la actuación de los robos en el simulador y generando nuevas estrategias de juego.

En este contexto, la gente del proyecto de los Robots Cuadrúpedos[34] ha creado un traductor de archivos. Los archivos ya programados con comportamientos específicos que utiliza el *XML Behaviour Control* pasan por dos procesos, el análisis y la evaluación. El proceso de análisis se genera por medio de la observación del agente en el simulador. Sólo si el agente realiza los estados y las acciones de manera correcta de acuerdo a su programación entonces pasa al segundo proceso. Se define la realización de acciones de manera correcta cuando se dan como entradas una serie de eventos que si suceden el agente tiene que obligatoriamente realizar una acción como salida del proceso.

El proceso de evaluación indica cuál es el mejor comportamiento programado para realizar una tarea, esto es, si se tienen varios comportamientos para conseguir un objetivo, entonces se selecciona sólo el que realice éste en el menor tiempo y con el menor número de estados posibles. Esto se debe a que es menos probable que el traductor falle en la traducción de los archivos si, éstos contienen menos estados. En el momento que se obtiene el mejor comportamiento, éste pasa por el traductor conocido como *TecRams Parser*. El traductor cambia los archivos del formato del *XML Behaviour Control* al formato del *XABSL*[35] para que dichos comportamientos generados se puedan observar de manera física en los *Sony Aibo*[36] [Figura 22].

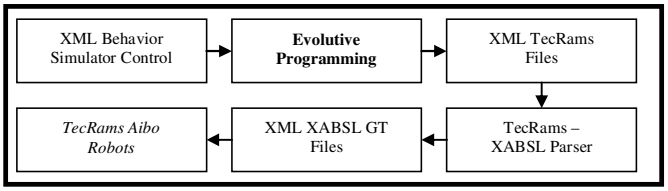


Figura 22. Proceso de generación de comportamientos utilizando el simulador XML Behaviour Control y el XABSL.

[34] Proyecto de los Robots Cuadrúpedos, 2006.
[35] XABSL: The Extensible Agent Behavior Specification Language. 2006.
[36] Sony Global- AIBO Global Link, 2006.

2. ANÁLISIS Y PROBLEMAS DE LOS APOYOS UTILIZADOS EN EL PROYECTO ROBOTS CUADRÚPEDOS.

Hasta este punto, se han introducido tres apoyos que utiliza *TecRams* para la generación de comportamientos: *XABSL*, *XML Behaviour Control*[37] y la herramienta basada en programación genética o evolutiva, *ABG*[38]. A continuación se muestra una introducción, ventajas y desventajas de cada uno de ellos, seguido de una solución temporal o parcial y finalmente el esquema general final para completar cada uno de los comportamientos en los *Sony AIBO*.

A. LA HERRAMIENTA XABSL.

El *Extensible Agent Behavior Specification Language*, conocido como *XABSL,* es un lenguaje muy simple que sirve para generar comportamientos en agentes autónomos. El *XABSL* fue desarrollado para diseñar comportamientos para agentes que juegan fútbol, sin embargo se pueden generar comportamientos para cualquier tipo de ambiente. De manera general, utiliza archivos en formato *XML* con un lenguaje específico que es interpretado y transformado por esta herramienta, que arroja como resultado, archivos en formato *C++*.

El problema fundamental que tiene esta herramienta radica precisamente en la generación de los comportamientos. El lenguaje que utilizan éstos, a pesar de ser sencillo, resulta complicado en primer lugar reconocer dónde hubo errores si es que los hay. Estos errores suelen ser muy comunes debido a que toda la programación es manual, esto es, el usuario o programador genera un comportamiento de acuerdo a lo que quiera que el agente haga y después lo compila. El *XABSL* es una herramienta poderosa para modificar los comportamientos de los agentes, mas no para generar comportamientos dinámicos en tiempo real [Ver anexo 8].

B. LA HERRAMIENTA *XML BEHAVIOUR CONTROL*.

El *XML Behaviour Control*[39] es un programa de simulación que mediante un lenguaje específico interpreta archivos en formato *XML* para generar comportamientos en un agente particular dentro del simulador. A pesar de que el lenguaje es aún más sencillo que el *XABSL*, el usuario tiene que necesariamente generar los archivos con los compartimientos requeridos, cosa que indudablemente provoca varias fallas.

La definición de comportamientos en *XML* resulta sencilla con la ayuda de herramientas disponibles para la manipulación de archivos *XML*. Además esta herramienta proporciona un modelo de definición de estrategias comprensible e independiente de la programación interna del robot. El resultado de esta simulación acelera el desarrollo tanto de nuevas estrategias como de

[37] Vega, et al. "Major behavior definition of football agents through XML" , p. 1-2.
[38] Tapia, et al. "Automatic behavior generation in a multi agent system through evolutionary programming", p. 2-3.
[39] Vega, et al. "Major behavior definition of football agents through XML" , p. 4.

acciones, o eventos percibidos por el robot, ya que sólo se necesita actualizar el listado de acciones disponibles y eventos para que un estratega sepa con qué nuevas funcionalidades cuenta.

Por otro lado, el modelado de un comportamiento por medio de máquinas de estado jerárquicas facilita la mejora continua y la reutilización de comportamientos, además se tiene la posibilidad de ir encapsulando comportamientos para que las máquinas de estado se mantengan en un nivel bajo de complejidad. Además el programa gráfico que tiene esta herramienta sirve para mostrar eventos, estados activos, inactivos y transiciones brinda un panorama más claro sobre el funcionamiento del autómata y facilita la evolución de las estrategias de los jugadores.

Con la adición del módulo de comunicación se probó la extensibilidad y flexibilidad del sistema para incorporar nuevas características. Este módulo facilita la creación de estrategias no sólo a nivel individual sino en conjunto, completando objetivos que han sido complicados en la liga RoboCup entre los que destacan el envío de pases y la posesión del balón.

C. LA HERRAMIENTA ABG.

El *Automatic behavior generation through evolutive programming*, conocido como *ABG*[40] es un sistema que utiliza las ventajas de la programación evolutiva para generar comportamientos de manera dinámica. El *ABG* fue desarrollado para trabajar sobre el *XML Behaviour Control* el cual utiliza varios métodos de la programación genética para finalmente arrojar un comportamiento de acuerdo a una serie de reglas que se tienen que seguir[41].

En primer lugar, y más importante, es la codificación de un cromosoma, que contiene todo un conjunto de comportamientos. Un comportamiento es la colección de varios estados con sus respectivas transiciones [Figura 23].

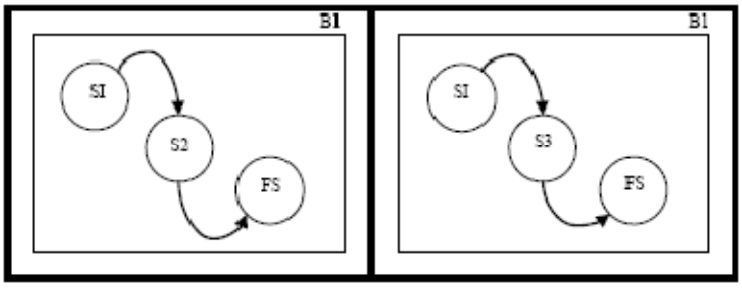


Figura 23. Generación de dos cromosomas con un comportamiento cada uno

La función de mutación cambia algún aspecto de un cromosoma, esto es, la mutación de acuerdo a una probabilidad dada por el sistema, es capaz de cambiar un comportamiento tanto de estado como de transición [Tabla 1].

[40] Tapia, et al. "Automatic behavior generation in a multi agent system through evolutionary programming", p. 2-3.
[41] Tapia, et al. "Automatic behavior generation in a multi agent system through evolutionary programming", p. 3-4.

Tabla 1. Probabilidades de Mutación.

Mutación	*Probabilidad*
Cambiar la salida de un estado	1.0
Cambiar la dirección de transición	1.0
Cambiar condición de transición	1.0
Agregar estado al comportamiento	0.5
Cambiar estado inicial	0.5
Borrar estado	0.5

La función de evaluación contiene las reglas necesarias y suficientes para controlar cada cromosoma por generación. Lo anterior se refiere que todos y cada uno de los cromosomas son evaluados por esta función de manera tal que de acuerdo a las reglas se va obteniendo una calificación. Por ejemplo, si se requiere de un comportamiento de un portero y si un cromosoma contiene como estado inicial la localización del agente dentro de la zona de portero y como estado final bloquear tiros, por lo tanto éste tendrá una mayor calificación que uno que tenga sólo tiros.

El problema de esta herramienta radica que genera un comportamiento ya esperado, esto es, cuando se programan las reglas para la evaluación de los cromosomas, se está limitando la búsqueda de manera que sólo cumplan ciertos requisitos, y eso al final genera un comportamiento esperado. Este comportamiento a pesar de que cumple con las necesidades del programador, desgraciadamente, está limitado precisamente en la búsqueda de los requisitos, de tal suerte que no se exploran todas las posibles opciones que se tienen para generar una solución de comportamientos para un problema específico.

Lo anterior significa que los comportamientos son creados a fuerza bruta, sin tener un orden concreto para la realización y creación de comportamientos, la mayoría como algoritmos genéticos utilizan toda la gama de posibilidades de acciones y con ciertos cambios por generación, son capaces de llegar a una solución que cumple con ciertos requisitos, pero jamás el agente sabe de donde vino esa solución, simplemente se le asigna y funciona.

D. SOLUCIONES TEMPORALES QUE SE UTILIZA EN EL PROYECTO DE LOS ROBOTS CUADRÚPEDOS.

Actualmente, el equipo *TecRams*[42] utiliza las herramientas anteriormente descritas de la siguiente manera. Primero, se obtienen los requerimientos específicos para un comportamiento en particular de un agente, por ejemplo, si se necesita un cambio de comportamiento en la salida del

[42] Proyecto de los Robots Cuadrúpedos, 2006.

portero ante una situación particular; después se colocan las reglas o funciones mínimas que son necesarias para la herramienta de la programación genética o evolutiva, al terminar se genera un archivo *xml* en el formato del simulador *XML Behaviour Control*[43] en el cual se analiza y verifica que haga el comportamiento, si no lo hace como se planeó desde un principio, entonces se modifica manualmente. Inmediatamente después de obtener dicho comportamiento simulado, este archivo *xml* es sometido a un cambio de formato (*traductor*), de tal suerte que se genera un nuevo archivo *xml* en el formato del *XABSL*[44] del *GermanTeam*[45] para que finalmente se someta a una compilación con todo el código de los módulos de *TecRams* y así tener finalmente el comportamiento en el campo de juego en un *Sony AIBO*[46] [Figura 24].

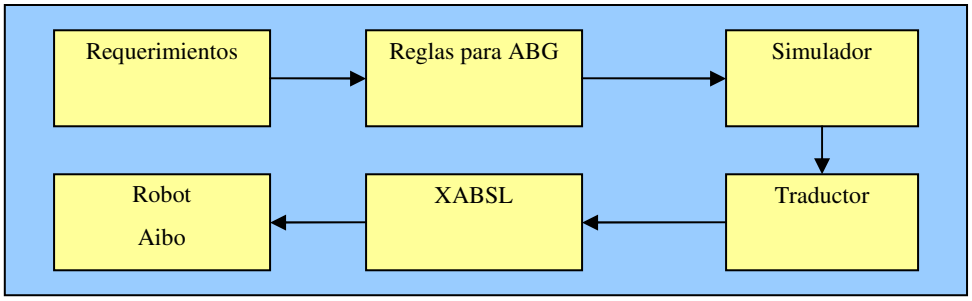


Figura 24. Proceso de generación de comportamientos utilizando el simulador XML Behaviour Control y el XABSL.

3. PLANTEAMINETO DEL PROBLEMA DE LA GENERACIÓN DE COMPORTAMIENTOS PARA TECRAMS.

Las soluciones actuales que realiza el proyecto de los Robots Cuadrúpedos[47] para la generación de comportamientos son por medio de la observación y del cambio inmediato de manera manual. Lo anterior significa que a pesar de tener un *ABG*[48], un simulador y un traductor, al final el equipo que integra el módulo de decisión tiene que obligatoriamente revisar cada instrucción en los archivos *xml* verificarlos y sobre todo corregirlos manualmente.

Lo anterior implica un problema de prueba y error, esto es, la probabilidad de fallo de generación de comportamientos es sumamente alta dado que es manual y a pesar de que se tiene un gran control, es muy común tener errores tanto sintácticos como semánticos. Definimos a un error sintáctico cuando un archivo no puede ser compilado debido a un error en la sintaxis del programa, y un error semántico se da cuando al ser compilado y ejecutado, el agente no realiza lo que se supone que debe de hacer.

[43] Vega, et al. “Major behavior definition of football agents through XML” , p. 4.
[44] XABSL: The Extensible Agent Behavior Specification Language. 2006.
[45] GermanTeam : Home, 2006.
[46] Sony Global- AIBO Global Link, 2006.
[47] Proyecto de los Robots Cuadrúpedos, 2006.
[48] Tapia, et al. “Automatic behavior generation in a multi agent system through evolutionary programming”, p. 2-3.

4. JUSTIFICACIÓN DEL USO DE APRENDIZAJE POR REFUERZO PARA LA GENERACIÓN DE COMPORTAMIENTOS.

La generación de comportamientos dentro del proyecto de *TecRams* es un problema que puede resolverse por medio del aprendizaje por refuerzo. Pues la hipótesis principal es que un agente sea capaz de generar sus comportamientos de acuerdo a las diferentes situaciones en el campo de juego, sin la necesidad de una supervisión humana.

En este modelo de aprendizaje, los agentes aprenden comportamientos por medio de interacciones basadas en ensayo y error, dentro de un medio dinámico. El aprendizaje ocurre a través de la experimentación basada en prueba y error con el medio ambiente.

La retroalimentación es en base a un pago escalar, esto es, mientras el agente se desenvuelve en su ambiente recibe una recompensa que es incremental y puede usarse en línea. Generalmente el tipo de tareas que realiza el agente bajo este esquema son reactivas.

Los algoritmos de aprendizaje por refuerzo están basados en:

1. Programación dinámica
2. Métodos Monte Carlo
3. Métodos de Diferencia Temporal

A. PROGRAMACIÓN DINÁMICA.

Éste método es de tipo divide y conquista. Se basa fundamental en el proceso de decisión de Markov (PDM). Programación dinámica es un algoritmo que recompensa por episodios el tamaño completo del problema, esto es, necesita del modelo completo del medio en el que va a interactuar cada agente. Generalmente este algoritmo conoce todas las recompensas y todos los episodios por lo que el agente obtiene dichas recompensas de acuerdo al episodio en el que esté circunscrito en ese momento.

Este tipo de algoritmos no fue utilizado para la realización de esta investigación, pues necesita de un modelo exacto y preciso del medio y un comportamiento se basa en acciones que no necesariamente generan la misma respuesta bajo condiciones ideales, además se necesita de un algoritmo que recompense de manera inteligente cada conjunto de acciones elegidas por el agente durante el tiempo y no esperar al final por ello.

B. MÉTODOS MONTE CARLO.

A diferencia del método de programación dinámica, los algoritmos de este método no necesitan de un modelo completo del medio, pues se basan fundamentalmente en la experiencia obtenida del agente cada vez que trata de interactuar con su ambiente.

Esta experiencia se genera por medio de tareas episódicas. Este algoritmo no recompensa por cada paso dado, sino que se espera a que se llegue al resultado final y a partir de ahí recompensa por cada episodio recorrido y almacenado en memoria.

Debido a que este método es adecuado para el cómputo incremental, no fue seleccionado para realizar las pruebas con los comportamientos de agentes de fútbol pues no se necesita almacenar mucha información en memoria por cada episodio recorrido.

C. MÉTODOS DE DIFERNCIA TEMPORAL.

Los métodos de diferencia temporal comparten las ventajas de los dos algoritmos anteriores: programación dinámica y Monte Carlo. Por una parte no necesitan conocer de todo el modelo del medio y si pueden ir recompensando episodio por episodio de acuerdo al nivel del modelo conocido.

Lo anterior se define de acuerdo a metas y estrategias predeterminadas para cada agente. Con esto, éste es capaz de generar un modelo del medio y de acuerdo a las estrategias determinar una recompensa inmediata para cada episodio. Es por ello que este tipo de métodos fue utilizado para desarrollar esta tesis, pues los comportamientos para el buen desenvolvimiento de agentes que juegan fútbol, no contienen todo el modelo del medio pues es dinámico y además puede tener recompensas inmediatas de acuerdo a los objetivos planteados para cada uno de los comportamientos a evaluar.

IV. *BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING* COMO HERRAMIENTA PARA LA GENERACIÓN DE COMPORTAMIENTOS DE AGENTES.

1. INTRODUCCIÓN A LA HERRAMIENTA BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING (*BADRL*).

Esta investigación utiliza la herramienta *XML Behaviour Control*[49] para generar una nueva solución de generación de comportamientos. La integración con esta herramienta resulta bastante útil pues su arquitectura computacionalmente es estable y sobre todo modular, esto es, resulta sencillo generar nuevas soluciones a problemas de decisión de agentes en este simulador [Figura 25].

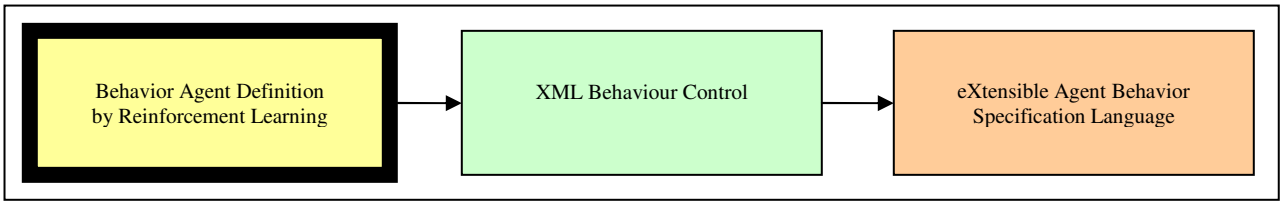


Figura 25. Esquema de la investigación utilizando aprendizaje por refuerzo con las actuales herramientas de *TecRams*.

Para lo anterior se define que el lenguaje de programación a utilizar es *Java* 5. El trabajo de investigación se hará en tres etapas principalmente: la generación de una nueva solución en el *XML Behaviour Control*, seguido de la aplicación del aprendizaje por refuerzo basado en el algoritmo *Q-Learning*[50] y finalmente la evaluación de los comportamientos simulados. Cabe aclarar que los resultados de esta investigación se mostrarán en archivos *xml* con formato del *XML Behaviour Control* y sólo se harán los cambios pertinentes en el traductor (*parser*) para que puedan ser interpretados por el *XABSL*[51] con los parámetros arrojados por esta investigación.

El nombre de la nueva herramienta creada es BADRL, *Behaviour Agent Definition by Reinforcement Learning*, el cual será modular para cualquier tipo de ambiente. Se define un sistema como modular en el instante en que será sencillo cambiar de solución para que los agentes puedan aprender las acciones que estén realizando sin importar lo que represente esa acción, pues no es lo mismo caminar hacia una pelota con cierta velocidad para jugar fútbol que caminar de manera dirigida para salir de un laberinto.

[49] Vega, et al. "Major behavior definition of football agents through XML" , p. 4.
[50] Sutton, et al. *Reinforcement Learning. An Introduction.* p. 56-60.
[51] XABSL: The Extensible Agent Behavior Specification Language. 2006.

En el caso anterior, son dos distintos ambientes sin embargo es una misma acción, la diferencia radica principalmente en la evaluación de dicha acción por ambiente, lo cual *BADRL* tendrá una configuración especial precisamente para agregar acciones y su evaluación. La configuración de las acciones y su evaluación se hará por medio de archivos con formato en *XML*. Lo anterior se definió debido a que éstos son evaluados mediante un *schema*.

2. CREACIÓN DE UNA NUEVA SOLUCIÓN EN EL SIMULADOR *XML BEHAVIOUR CONTROL*.

Como primer punto primordial, se utilizará el *XML Behaviour Control*[52] debido a que se puede evaluar sus resultados mediante dos medios: simulado y físico. Esta herramienta permite generar comportamientos y observarlos en simulación, además se tiene un *parser* para colocar el código simulado directamente en los *Sony* AIBO[53]. Dado lo anterior, se generará un nuevo paquete de código sobre éste, el cual tendrá como entradas en una clase especial, las acciones y recompensas. Estas entradas tendrán acceso a la solución la cual tendrá como resultado un conjunto de acciones aprendidas, las cuales pasarán por un proceso de generación de comportamientos únicamente con esas acciones [Figura 26].

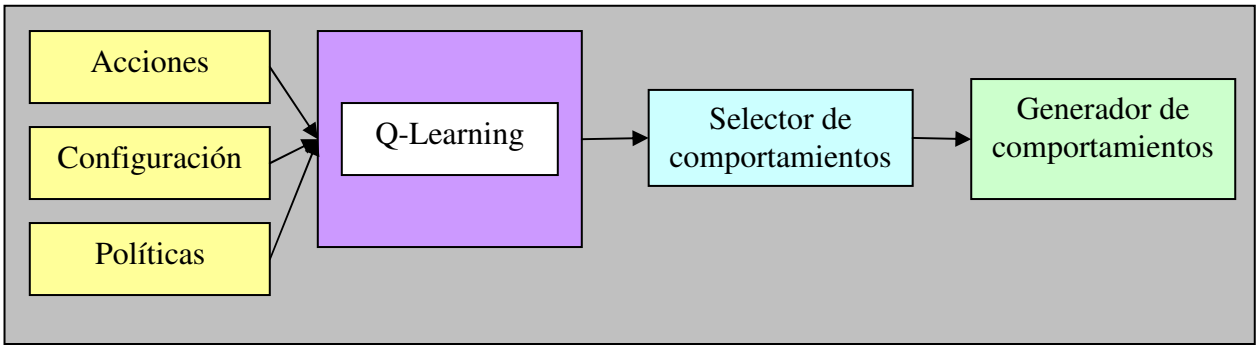


Figura 26. Esquema de la solución por agregar en el XML Behaviour Control.

A. IMPLEMENTACIÓN DE BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING.

El *BADRL* se puede ver como una caja negra configurable, la cual tendrá como entrada un conjunto de acciones y recompensas. Esta caja será capaz de evaluar cada una de las entradas y las evaluará completamente todas y cada una de ellas. Ésta tendrá la capacidad de integrarse con problemas modulares que puedan resolverse mediante aprendizaje por refuerzo. Además se podrá obtener un conjunto de reportes los cuales tendrán estadísticas de las evaluaciones cada vez que un agente ha ido aprendiendo una acción.

[52] Vega, et al. "Major behavior definition of football agents through XML" , p. 4.
[53] Sony Global- AIBO Global Link, 2006.

El conjunto de entradas serán evaluadas por la caja negra primeramente, después entraran a un proceso de ejecución mediante el algoritmo de *Q-Learning*. Finalmente se generará un objeto especial que contenga el conjunto de acciones aprendidas para poder ser utilizadas para generar algún comportamiento específico [Figura 27].

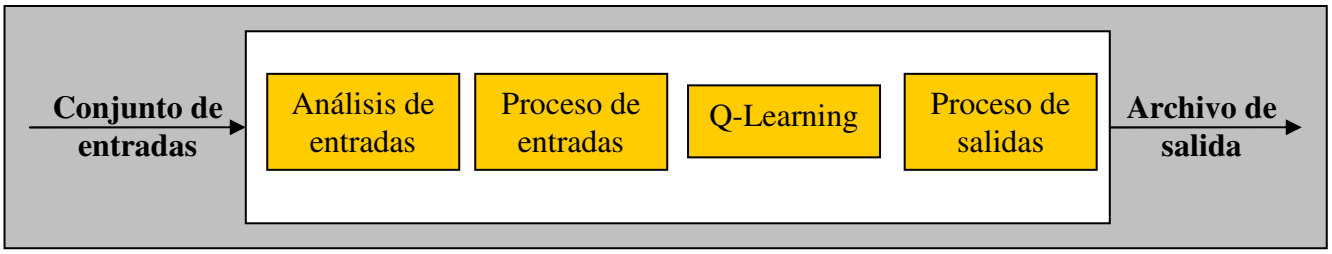


Figura 27. Procesos generales de *BADRL*.

La implementación de *BADRL* fue planeada y realizada con base al código de desarrollo conocido como Free Connectionist Q-Learning Java Framework [19]. Éste es un framework que utiliza y necesita principalmente tres configuraciones o implementaciones: una clase (*MyPerception*) que perciba el ambiente por medio de un arreglo de sensores y que tenga la función de recompensa para la evaluación de la acción elegida por el robot, una clase robot que mande a llamar la clase cerebro (*Brain*) con la información de los parámetros para que Q-Learning pueda funcionar y finalmente un arreglo de acciones donde cada acción es una clase que hereda de *Action* del ambiente de trabajo. La figura 28 muestra la relación entre las tres entidades descritas anteriormente dentro del Free Connectionist Q-Learning Java Framework.

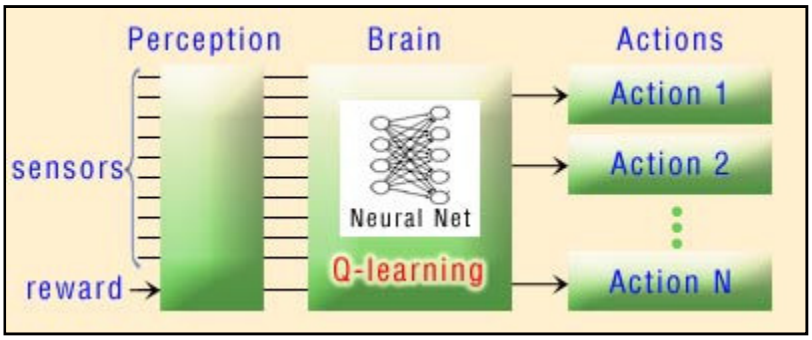


Figura 28. Relación de clases dentro de *Free Connectionist Q-Learning Java Framework*.

En el momento en que se implementó este framework para generar comportamientos, *BADRL* fue planeado y diseñado en cinco fases primordialmente:

1. Generación de archivo de configuración con la información sobre los parámetros que determinan el control del cerebro de la aplicación (*Brain*).
2. Generación de archivo de configuración con las acciones que puede realizar el agente y que cada una de éstas se traduce automáticamente en una clase anónima heredando de la clase *Action*.

3. Generación de comportamientos utilizando la información arrojada por el sistema Free Connectionist Q-Learning Java Framework para cada acción seleccionada.
4. Generación de archivo de configuración con las reglas y la evaluación de cada comportamiento definido anteriormente.
5. Generación del reporte estadístico con la información monitoreada por *BADRL*.

B. GENERACIÓN DEL REPOSITORIO DE ACCIONES PARA *BADRL*.

Las acciones son generadas automáticamente al leer el archivo *actions.xml* creado por el programador con el fin de crear un repositorio de acciones. Éstas son generadas por medio de clases anónimas para crear un arreglo que es utilizado por el Free Connectionist Q-Learning Java Framework (*Connectionist*). Connectionist selecciona una acción y ésta en el momento de ser ejecutada genera una transición a un estado con una acción atómica y al terminar éste le regresa al estado principal que lo mandó a llamar. Lo anterior se explica debido a que en primera instancia, BADRL genera por cada acción programada en *actions.xml* un estado con la acción atómica que se requiere más una transición a un estado principal [Figura 29].

```
<!-- DEBUG XMLWriting: ACTION STATES -->
<state name='camina_lento' iterate='true'>
      <action id='MOTION-goForward' param1='2'/>
      <transition state='firstState'/>
</state>
```

Figura 29. Estado que manda a llamar la acción atómica de caminar lento y una transición al primer estado que lo mandó a llamar.

Después, se genera el estado principal cuya acción no realiza nada más que esperar eventos que sucedan del ambiente para utilizar esas acciones generadas anteriormente [Figura 30].

```
<state name='firstState' iterate='true'>
      <action id='MOTION-doNothing'/>
      <transition state='tira_derecha_leve'>
            <event name="VISION-shootDistance" />
            <event name="VISION-ballAtLeft" />
      </transition>
      <transition state='tira_izquierda'>
            <event name="VISION-shootDistance" operator="not" />
            <event name="VISION-ballAtRight" />
      </transition>
      <transition state='tira_izquierda'>
            <event name="VISION-shootDistance" operator="not" />
            <event name="VISION-ballAtRight" />
      </transition>
      <transition state='tira_izquierda'>
            <event name="VISION-shootDistance" operator="not" />
            <event name="VISION-ballAtRight" />
      </transition>
</state>
```

Figura 30. Generación de transiciones del primer estado por BADRL a los estados con acciones atómicas.

C. LECTURA DE PARÁMETROS PARA Q-LEARNING GRACIAS AL ARCHIVO DE CONFIGURACIÓN DE *BADRL*.

Q-Learning es un algoritmo que utiliza como modelo aprendizaje por refuerzo con diferencia temporal. Para lo anterior, Q-Learning se basa en la siguiente fórmula [Fórmula 1]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a [Q(s_{t+1}, a_t)] - Q(s_t, a_t)\right]$$

Fórmula 1- Definición matemática del algoritmo Q-Learning.

Donde s_t es el estado en el que se encuentra el agente y realiza la acción a_t. $Q(s_t, a_t)$ es el valor y no recompensa, obtenido de estar en estado s_t y realizar la acción a_t.

Como se puede ver en la fórmula anterior, existen dos parámetros que determinan el grado de aprendizaje de la fórmula y éstos son: α (alfa) y γ (gamma).

Alfa es conocido como el parámetro que indica el tamaño del paso de aprendizaje, esto es, el grado de aprendizaje de lo desarrollado o vivido anteriormente por el agente. Es un valor probabilístico entre 0 y 1 al igual que Gamma. Éste es un parámetro que indica qué tanto el algoritmo utiliza el máximo de los valores Q para el próximo estado dada la acción anterior. Una manera de visualizar la fórmula general anterior con este parámetro se ejemplifica en la Figura 31.

$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right]$$

Figura 31. Parámetro alfa como tamaño del paso en la fórmula general de Q-Learning.

En el sistema BADRL, estos parámetros se encuentran definidos en el archivo *conf.xml*. En este archivo se pueden configurar estos parámetros dentro de la etiqueta *params.* Esto con el fin de que el usuario coloque los valores pertinentes de acuerdo a la meta a seguir por el agente. [Figura 32]

```
<!--  qLearning parameters  -->
      <parameters>
            <alfa-param value="0.5"/>
            <gamma-param value="0.8"/>
            <lambda-param value="1"/>
            <boltzmann-param value="true"/>
            <temperature-param value="0.9"/>
      </parameters>
```

Figura 32. Uso del archivo *conf.xml* para modificar los variables utilizadas por Q-Learning.

D. DEFINICIÓN DE METAS Y OBJETIVOS PARA LA GENERACIÓN DE COMPORTAMIENTOS EN *BADRL*.

Una meta es considerada como el comportamiento ideal donde un agente es capaz de realizar una tarea específica. El cumplimiento de la tarea denota un comportamiento legible, coherente y eficaz que puede hacer un agente. Se denota como legible a la característica del comportamiento cuando el usuario puede entender cada acción que ejecuta el agente en su comportamiento, coherente como una manera estructurada que sea capaz de realizar el agente y por último eficaz, esto es, que haga lo que dice hacer.

Dentro del sistema *BADRL*, las metas se encuentran descritas dentro del archivo *rules.xml*. Este archivo fue generado con el propósito de que el usuario sea capaz de cambiar las reglas y/o políticas de evaluación que seguirá el agente durante la ejecución del desarrollo del comportamiento. Las reglas son evaluadas de acuerdo al siguiente órden:

1. Cada una de las reglas es evaluada de manera secuencial.
2. Cuando una regla no cumple con todas las características que tiene del modelo actual del medio entonces, la evaluación seguirá con la regla posterior.
3. Cuando una regla cumple con todas las características dentro del modelo y le toca ser evaluada, en ese momento se disparan las recompensas y se cierra el comportamiento.
4. Cuando se acaban las reglas y no hay ninguna que se haya cumplido, el sistema volverá a leer y evaluar cada una de las reglas de manera secuencial desde el inicio.

Dado lo anterior, la meta está definida como la primera regla dentro del archivo, pues es la primera a evaluarse si el agente ya cumplió con el objetivo planteado. Como ejemplo, en el caso concreto de la conducción de la pelota, se definió como meta si el agente avanza con la pelota dominada más de la mitad del campo.

Se entiende como dominada cuando el agente es capaz de tener la pelota, y además éste es capaz de ubicarla dentro de la cancha y éste se encuentra al esférico a una distancia tan cerca que puede patearla [Figura 33].

```
<ql-rule id="1" name="PRINCIPAL_TARGET" useTimer="18">
    <and>
        <ql-event name="eventSD" value="VISION-shootDistance" function="isActive()"/>
        <ql-event name="eventBF" value="VISION-ballFound" function="isActive()"/>
        <ql-event name="agent" value="positionX" function="greaterThan(80)"/>
    </and>
    <reward goWell="true" goBad="false" />
    <debug text="Target Reached!!" info="positionX" />
</ql-rule>
```

Figura 33. **Definición de la meta por medio de la primera regla de *rules.xml* para la conducción de la pelota.**

E. DEFINICIÓN DE POLÍTICAS Y LA FUNCIÓN DE RETROALIMENTACIÓN PARA BADRL.

Un criterio de evaluación es el conjunto de eventos disparados después de ejecutar un comportamiento en un tiempo finito. Es considerado que el criterio más importante de evaluación es la meta que persigue el agente. Para el caso de la conducción se definió la meta de la siguiente manera [Figura 34]:

1. Si el sistema ya corrió por más de veinte segundos.
2. Si el agente tiene la pelota lo suficientemente cerca para patearla.
3. Si el agente ve la pelota.
4. Si el agente se encuentra en una posición cercana a la portería.

```
if( nextRun.getTime() - firstRun.getTime() > 20000  &&
      eventSD.isActive() && eventBF.isActive() &&
      aibo.getPhysicObject().getPosition().getX() >= 80
    )
```

Figura 34. Criterio de evaluación para determinar si un agente ha logrado aprender la conducción de la bola.

Por otro lado, se definieron los siguientes criterios para determinar si un agente va realizando acciones coherentes o no:

1. Si pasan más de veinte segundos y
    a. El agente ve la bola pero no llega cerca de la portería contraria.
    b. El agente no ve la bola.
2. Si encuentra la bola.

Hasta este punto es necesario señalar que la definición de criterios de evaluación es importante para el buen desempeño del sistema BADRL. Fue por ello que se generó el archivo *rules.xml* descrito en este capítulo en el tema anterior. Para cada comportamiento que se requiera, es necesario cambiar este archivo con la intención de abarcar los posibles problemas y aciertos que llegara a tener el agente durante su aprendizaje. Es claro que si se generan más reglas, será más nítido el proceso de aprendizaje pues se abarcará más campo de calificación para los estados y acciones que genere el agente.

Ejemplo de lo anterior se puede comprobar con la generación del comportamiento de conducción de la pelota, pues se utilizaron 5 reglas principalmente:

1. Cuando el agente llega a la meta.
2. Cuando avanza con la pelota un tramo y tiene la pelota en su posesión pero no llega a la meta.
3. Cuando avanza con la pelota pero pierde la pelota de vista.
4. Cuando avanza muy poco y alcanza a ver la pelota.
5. Cuando no avanza a pesar de que vea la pelota.

La figura 35 es un ejemplo de la definición de las primeras tres políticas para la generación del comportamiento de la conducción.

```
    <and>
        <ql-event name="eventSD" value="VISION-shootDistance" function="isActive()"/>
        <ql-event name="eventBF" value="VISION-ballFound" function="isActive()"/>
        <ql-event name="agent" value="positionX" function="greaterThan(80)"/>
    </and>
    <reward goWell="true" goBad="false" />
    <debug text="Target Reached!!" info="positionX" />
</ql-rule>

<ql-rule id="2" name="OVERTIME and position greater and ball found" useTimer="18">
    <and>
        <ql-event name="agent" value="positionX" function="greaterThan(0)"/>
        <ql-event name="eventBF" value="VISION-ballFound" function="isActive()"/>
    </and>
    <reward goWell="true" goBad="false"/>
    <debug text="Position = " info="positionX" />
    <debug text="Time is Over: reward position + ballFound"/>
</ql-rule>


<ql-rule id="3" name="OVERTIME and position greater" useTimer="18">
    <ql-event name="agent" value="positionX" function="greaterThan(0)"/>
    <reward goWell="true" goBad="false"/>
    <debug text="Position = " info="positionX" />
    <debug text="Time is Over: reward position + ballFound"/>
</ql-rule>
```

Figura 35. Archivo *rules.xml* con la descripción de las primeras 3 reglas para la conducción de la pelota.

3. PROCESO DE GENERACIÓN DE COMPORTAMIENTOS QUE USA BADRL.

Este proceso es la base primordial de todo el sistema BADRL y el motivo de estudio y evaluación que comprende esta tesis. La generación de comportamientos utilizando el algoritmo *Q-Learning* se debe de ver de la siguiente manera: el usuario utiliza los 3 archivos de configuración para colocar acciones, reglas, parámetros y posible retroalimentación para el sistema, después, dentro de un ciclo de *n* iteraciones, donde *n* es un número definido por el usuario dentro del archivo de configuración *actions.xml*, el sistema *BADRL* manda a llamar el cerebro (*Brain*) del framework para que seleccione de acuerdo a las características de Q-Learning la acción que corresponda. Es importante señalar un par de cosas:

1. Si el comportamiento deseado es de *n* acciones, y el número de la acción utilizada por Q-Learning es menor a esa cantidad, *BADRL* utilizará del archivo de configuración *conf.xml*

la recompensa por default para retroalimentar a Q-Learning con dicha calificación a la acción seleccionada.

2. Si la acción utilizada por *Q-Learning* es el mismo número de iteraciones *n* por comportamiento deseado, entonces se genera un comportamiento en formato *XML* que es entendido e interpretado por el sistema *XML Behaviour Control*. Este comportamiento es guardado en la carpeta *finalBehavior* del sistema *BADRL* e inmediatamente se manda a llamar la clase que evalúa dicho comportamiento de acuerdo al ambiente simulado generado. Si el comportamiento consigue la meta final entonces el proceso para, de lo contrario se genera un reporte con la nueva información de ese comportamiento y su evaluación y se inicializa el contador de acciones en cero para que se vuelva a generar otro comportamiento y regrese al punto 1. En la figura 36 se puede observar el código principal de BADRL que permite llevar a cabo sus principales actividades: crear acciones, evaluar acciones, generar comportamiento y evaluar comportamiento.

```
while ( ! targetReached ){
      //Initialice counter
      count = 0;

      //Getting and evaluating actions that form one behavior
      while ( count < numberOfTransitions ){
            robot.goBad = false;
            robot.goWell = false;
            robot.count();
            count++;
      }

      //Generating the environment
      xmlExec = new GenerateEnvironment(envFileName, fileName);
      xmlExec.generateEnvironment();

      //Generating the behavior to evaluate
      generateBADRLFile();

      //Evaluating the behavior created
      evaluator = new Evaluator(ruleSet, envFileName, null, args,
          robot, this.visible);
      evaluator.start();
      semaforo = true;

      while(semaforo){
            try Thread.sleep(1000);
            catch (InterruptedException e) e.printStackTrace();
      }

      //Generating the evaluation report of the behavior created
      generateReport(runs++);
}
```

Figura 36. Proceso general de la generación de comportamientos de *BADRL*.

A. GENERACIÓN DEL AMBIENTE SIMULADO PARA EL *XML BEHAVIOUR CONTROL.*

En este punto es importante señalar que cada comportamiento deseado requiere de un ambiente personalizado para su correcta evaluación y aseguramiento de un buen aprendizaje de una actividad en particular del agente.

BADRL utiliza el formato del ambiente que lo mande a llamar, en este caso, se programó que generara una copia del ambiente del *XML Behaviour Control* llamado *environment.xml* el cual tiene que ser modificado por el usuario cada vez que quiera cambiar de escenario para la evaluación de sus comportamientos [Figura 37].

```
<environment virtual_space="PlayGround2007" robot_delay="40"
      object_delay="150" friction="120">

      <!-- Approach Players -->
      <robot type="aibo" file="rl/finalBehavior/manualApproaching.xml"
            team="BLUE"       name="CuauBADRL"  initial_positionX="-400"
            initial_positionY="200"       position="LEFTATACKER"/>

      <!--  The others -->
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Seis"   initial_positionX="20"  initial_positionY="-40"
            position="DINAMICATACKER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Cinco" initial_positionX="20"   initial_positionY="40"
            position="DINAMICDEFENDER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Cuatro" initial_positionX="-20" initial_positionY="0"
            position="CENTERATACKER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Tres"  initial_positionX="-20"  initial_positionY="40"
            position="RIGHTDEFENDER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Dos"   initial_positionX="-20"  initial_positionY="-40"
            position="LEFTDEFENDER"/>
      <robot type="aibo" file="keeper2.xml" team="RED"
            name="Tala"  initial_positionX="200"  initial_positionY="0"
            position="DINAMICKEEPER"/>
</environment>
```

Figura 37. Ambiente especializado para una evaluación correcta del comportamiento *approachToBall.*

B. SELECCIÓN DE ACCIONES POR MEDIO DE Q-LEARNING.

Para poder explicar de mejor manera esta sección, la selección de acciones se puede dividir en dos partes fundamentalmente: la sección utilizada por el algoritmo Q-Learning dentro del esquema de BADRL y la sección de redes neuronales utilizada por el algoritmo Q-Learning. Estaremos refiriéndonos a la primera sección como acciones en BADRL y la segunda como acciones en Q-Learning.

Como inicio, es importante mencionar cómo se obtienen las acciones en el sistema BADRL. Las acciones son creadas por el programador mediante un archivo conocido como *actions.xml*. Este archivo contiene el conjunto de acciones posibles que puede utilizar el agente en un comportamiento. Estas acciones pueden ser acciones atómicas, acciones complejas o un comportamiento específico.

Sucesivo a que el sistema BADRL lee estas acciones, genera una lista de estos objetos y los introduce al sistema de Q-Learning. Después, este algoritmo se activa después de que todos los sensores y actuadores del agente físico o simulado estén conectados a este sistema. Es justamente en este momento cuando funciona el cerebro de esta tesis.

El algoritmo Q-Learning es la principal herramienta para la selección de las acciones. Cuando las acciones son introducidas, se utilizó una red neuronal para la propagación de los valores Q de cada una de estas acciones. Lo anterior significa, que los pesos de cada una de las acciones son mantenidas por una red neuronal, pero la decisión de cual tomar es de Q-Learning mediante su fórmula de aprendizaje por refuerzo.

Lo anterior es muy importante aclarar, pues para elegir una acción, en primer lugar Q-Learning propaga la red neuronal, seguido de recorrer cada uno de los valores asignados a cada acción y completar la información de la fórmula para finalmente tomar la decisión de la acción. En la figura 38 se puede analizar precisamente esta función primordial para el algoritmo.

```
/**
 * Selects an action to execute, basing on the values of Q-function.
 * @return  number of the selected action
 */
private int selectAction() {
    int a = -1;
    Qmax = -1;
    propagate();
    for (int i = 0; i < Q.length; i++) {
        if(useBoltzmann) {
            boltzValues[i] = Math.exp(Q[i]/temperature);
        }
        if(Qmax < Q[i]) {
            Qmax = Q[i];
            a = i;
        }
    }
    if(useBoltzmann) {
        a = RR.pickBestIndex(boltzValues);
    }
    return a;
}
```

Figura 38. Código de la función principal de Q-Learning para seleccionar una acción.

Como se puede visualizar en la figura anterior, la selección de la acción depende de los pesos brindados en la función *propagate()* de la red neuronal de Q-Learning, y a partir de ahí de acuerdo al uso del método de Boltzmann y la temperatura configurada, se obtiene el mayor de los valores Q y el método selectAction() regresa la acción que cumple con dichas características. Sin embargo, en ninguna parte del código se utiliza la fórmula clásica de Q-Learning. Es en este punto en el cual se tiene que poner especial atención, pues esta función es privada y por lo tanto no puede ser llamada por ninguna instancia de otra clase.

En realidad lo que sucede aquí, es que esta función es llamada como primera rutina en otra función llamada *count()*, la cual se encarga de realizar el proceso de la selección de la acción seguida de los cambios y actualizaciones en la red neuronal utilizando la regla clásica de Q-Learning.

En la figura 39 se puede visualizar el contexto del método *count()*. En primer lugar se tiene la llamada a la función *selectAction()* que nos regresa la acción con el máximo valor Q después de propagar la tabla, después se ejecuta la acción y se obtiene la recompensa inmediata del medio. Recordemos que esta recompensa se obtiene mediante el uso de las reglas contenidas en *rules.xml* y los parámetros de configuración de *conf.xml*. Después esta recompensa es utilizada para generar el margen de error y actualizar los pesos. Después se vuelve a propagar la red neuronal con las nuevas actualizaciones y se guarda en memoria el valor Q obtenido anteriormente, para que en la próxima iteración se utilice como un valor Q previo (Qprev).

```
/**
 * One step of the Q-learning algorithm. Should be invoked at every time step.
 * It is responsible for selecting the action and updating weights.
 * DOES NOT execute any action. For this use Brain.execute() method.
 * @see Brain#executeAction()
 */
public void count() {
    a = selectAction();
    if(tactCounter > 0) {
        double r = perception.getReward();      // r(t-1)
        double error = r+gamma*Qmax-QPrev;
        updateWeights(error);       // w(t)
    }
    propagate();
    countEligibilities(a);      // e(t), g(t)
    tactCounter++;
    QPrev = Q[a];
}
```

Figura 39. Código del método público count() de Q-Learning para seleccionar una acción.

Finalmente, ya que el método *count()* nos regresa la acción y actualiza los valores de la red neuronal, es importante señalar cómo la red neuronal los actualiza de acuerdo a la fórmula de Q-Learning. Si bien se explicó que el margen de error, segunda parte de la fórmula de Q-Learning, se utiliza para actualizar la red, falta analizar la primera parte de esta fórmula. Para ello se explicará el método *updateWeights()*.

El método *updateWeights()* es un método privado que sólo es llamado por *count()* cuya función es actualizar los valores de la red neuronal utilizando la fórmula de Q-Learning. Para ello es necesario introducir el valor del margen de error y a partir de ahí generar la primera parte de la fórmula de Q-Learning.

En la figura 40 se puede observar que el *updateWeights()* recorre cada uno de los valores de la red neuronal para actualizar los valores. Para ello, utiliza el valor del margen de error, ahora conocido como cambio y se multiplica el valor de alfa, valor obtenido del archivo de configuraciones *conf.xml*, junto con el peso anterior. Es importante señalar que aquí es donde realmente se entiende claramente como se maneja la red neuronal y Q-Learning. Pues la red sólo se utiliza para almacenar estos valores Q y Q-Learning se encarga de percibir del ambiente, de actualizar los valores Q con la fórmula clásica y además de tomar el máximo valor Q mediante el uso del algoritmo de Boltzmann.

```
/**
 * Used to teach the neural network. Updates all the weights
 * basing on eligibility traces and the change value.
 * @param change
 */
private void updateWeights(double change) {
    for (int l = w.length-1; l >= 0; l--) {
        for (int i = 0; i < w[l].length; i++) {
            for (int j = 0; j < w[l][i].length; j++) {
                w[l][i][j] = w[l][i][j] + (alpha * change * e[l][i][j]);
            }
        }
    }
}
```

Figura 40. Código del método privado *updateWeights()* de Q-Learning para seleccionar una acción.

Es importante señalar que la investigación de esta tesis es utilizar este algoritmo como parte fundamental para generar acciones y completar comportamientos para agentes de fútbol. El hecho de que este algoritmo utilice una red neuronal para actualizar los valores Q no es primordial en el análisis de esta investigación. Los resultados se verán afectados únicamente por la cantidad de acciones utilizadas, tipo de comportamiento y valores utilizados en el archivo de configuración *conf.xml* no por las capas ni neuronas utilizadas en la red neuronal.

C. USO DE LAS METAS Y OBJETIVOS PARA GENERAR COMPORTAMIENTOS.

Una vez que Q-Learning es capaz de arrojar una acción para que el agente logre realizar un movimiento en el ambiente, es importante señalar que pasa después con el trabajo de investigación de esta tesis. Ya que el agente recibe qué acción realizar si no llega a cumplir con

un comportamiento, ésta acción recibe la recompensa por default configurada en *conf.xml*. Para que esta recompensa pueda darse al agente tendrá que necesariamente cumplir con alguna regla definida en *rules.xml*.

Ahora bien, ya que se tiene entendido el esquema general para dar recompensa de acuerdo a las reglas definidas, sigue la parte de la definición de metas y objetivos que el agente debe seguir para completar su aprendizaje para un comportamiento específico. La meta u objetivo se tiene que definir forzosamente en el archivo *rules.xml*. Esta meta tiene que ser escrita como primera regla fundamental en el archivo. Es el único requisito que se tiene para configurarse en BADRL. Un ejemplo de ello es la figura 41, la cual nos muestra que el objetivo es que el agente tiene que llegar a la bola a una distancia que éste pueda patear, que éste pueda verla sin ningún obstáculo en medio y que la posición del agente esté mayor del cuarto de cancha. Esto es claro que lo que se necesita es que el agente se acerque a la bola y no la pierda por ningún motivo. Cuando sucede esto generalmente se coloca como texto en la etiqueta debug, alguna información que le indique al programador que el agente a alcanzado el objetivo planteado.

```
  <!--   TARGET   -->
- <ql-rule id="1" name="OBJECTIVE_TARGET" useTimer="20000">
  - <and>
      <ql-event name="eventSD" value="VISION-shootDistance" function="isActive()" />
      <ql-event name="eventBF" value="VISION-ballFound" function="isActive()" />
      <ql-event name="agent" value="positionX" function="greaterThan(-30)" />
    </and>
    <reward goWell="true" goBad="false" />
    <debug text="Target Reached!!" info="positionX" />
  </ql-rule>
```

Figura 41. Recompensas para los criterios de evaluación para BADRL.

D. EJECUCIÓN DE LOS CRITERIOS DE EVALUACIÓN Y RETROALIMENTACIÓN PARA Q-LEARNING.

Para que un criterio de evaluación cumpla su objetivo, es necesario y obligatorio regresar una retroalimentación para *Connectionist*. Esta retroalimentación es creada por el programador en el archivo *conf.xml* dentro de las etiquetas rewards [Figura 42].

```
<rewards>
      <movingWell-reward value="-0.05" />
      <collides-reward value="-0.2"/>
      <default-reward value="-0.2" />
</rewards>
```

Figura 42. Recompensas para los criterios de evaluación para BADRL.

Actualmente estas recompensas se utilizan como banderas que se activan de acuerdo a lo que sucede en los criterios de evaluación.

Por ejemplo para el caso de la conducción, si el agente pierde la bola después de su periodo entonces se activa la bandera de collides-reward y se apaga la de movingWell-reward, de manera tal que BADRL obtenga la retroalimentación correcta [Figura 43].

```
if( ! getEventByName(eventBallFound).isActive())
      robot.goBad = true;
robot.goWell = false;
}
```

Figura 43. Forma de activar las banderas para retroalimentar a Q-Learning con las recompensas

E. GENERACIÓN DE REPORTE ESTADÍSTICO CON LA INFORMACIÓN ARROJADA POR BADRL.

En general, se pretende que *BADRL* tenga la capacidad de generar reportes estadísticos de manera simple pero con información importante para el programador. Además se pretende generar archivos en *html* que contengan la información de evaluación de las acciones respecto al tiempo de cada corrida de cada uno de los algoritmos con el fin de determinar hasta que momento el agente fue capaz de aprender a su máxima capacidad las acciones que *BADRL* tuvo como entrada.

La generación de comportamientos para un medio dinámico de agentes de fútbol es un proceso que involucra varias iteraciones para llegar a las metas y objetivos planteados por el programador. Para ello, BADRL fue generado para tener dos facetas primordialmente:

1. La opción de mostrar al usuario la interfaz gráfica con los comportamientos de cada iteración para el agente.
2. La opción de correr BADRL sin la interfaz y sólo mostrar al usuario algunos detalles para tener la seguridad de que no ha existido errores en el transcurso de las iteraciones.

Para las dos opciones, BADRL genera automáticamente un reporte que contiene todas las actividades por iteración del agente. Dichas actividades son:

1. Número de iteración
2. Arreglo generado de acciones para crear un comportamiento.
3. La recompensa inmediata otorgada.
4. El promedio que en ese momento tiene Connectionist.
5. Finalmente un resumen de lo que sucedió al correr ese comportamiento [Figura 44].

BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING

Report File

Initial Time: *Wed Apr 11 20:14:30 CDT 2007* **Final Time:** *Wed Apr 11 20:21:08 CDT 2007*

QL Iteration Number	Selected Actions	Reward	Average	Debug
0	• camina_lento • tira_loco_frente • tira_loco_frente • tira_leve_frente • tira_leve_frente • tira_derecha_leve • tira_derecha_leve • camina_lento	-0.05	-0.015451061114415979	Position = (54.0 , 0.0) Time is over: reward position + ballFound
1	• camina_lento • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • camina_lento • camina_lento	-0.2	-0.02970844578102487	Position = (-18.0 , 0.0) Time is over

Figura 44. Reporte final generado por BADRL para la creación del comportamiento de la conducción.

V. RESULTADOS Y EVALUACIONES DE LOS COMPORTAMIENTOS GENERADOS POR *BADRL*.

Este capítulo trata sobre los resultados arrojados por la herramienta *BADRL*. Se entiende como resultado a la evaluación completa del proceso de aprendizaje. Es importante definir que el comportamiento final aprendido es parte de este proceso. Cada iteración generada en *BADRL* cambia o regenera un archivo final que se conoce como reporte.

El análisis que se realiza en este capítulo se enfoca en los reportes generados por *BADRL* para cada comportamiento diferente. Esta tesis se enfocó principalmente en el estudio de cuatro casos:

1. Conducción de la bola.
2. Búsqueda y alineación del robot con la bola.
3. Aproximación del robot hacia la bola.
4. Tiro del robot con la bola hacia la portería.

Además se muestra en cada uno de estos temas un análisis comparativo respecto al trabajo realizado con algoritmos genéticos. Lo cual ayuda a entender estos dos temas distintos y que llegan a tener resultados muy diferentes. Finalmente se arrojan los resultados generales de todos los comportamientos y la evaluación final de la herramienta *BADRL*.

1. INTRODUCCIÓN A LAS EVALUACIONES DE LOS COMPORTAMIENTOS.

El archivo final o reporte contiene el comportamiento aprendido durante la iteración actual del sistema de aprendizaje de *BADRL*. En el momento en que se genera una nueva iteración, el archivo cambia para contener el nuevo comportamiento. Como se puede ver, el problema de lo anterior radica que se pierde la referencia del comportamiento en cada iteración. Para ello, se genera en cada una de éstas, un archivo en formato *HTML* que contiene 7 partes principalmente:

1. **Tiempos de corrida (inicial y final).**

En la figura 45 se muestra un ejemplo de los tiempos de corrida. Un tiempo de corrida se define como el tiempo que se tarda *BADRL* en generar un comportamiento que cumpla con los requisitos definidos en la primera regla de la meta u objetivo. El archivo de resultados *BADRL* contiene la fecha con la hora, minuto y segundo con el que empezó y el tiempo en el que acabó de generar el aprendizaje.

Figura 45. Tiempo inicial y final del archivo HTML con los resultados finales de BADRL.

2. Conjunto de acciones.

El conjunto de acciones se define en el archivo *actions.xml*. De este archivo se sacan los nombres de las acciones las cuales ayudan a visualizar en cada iteración cuáles son las posibles acciones que puede utilizar *BADRL* para generar el comportamiento. En la figura 46 se muestra un ejemplo de un conjunto de cuatro acciones que maneja *BADRL*.

Figura 46. Tiempo inicial y final del archivo HTML con los resultados finales de BADRL.

3. Número de la iteración de *BADRL*.

El número de la iteración ayuda a llevar el control de los comportamientos generados por *BADRL*. Se puede determinar en cuáles momentos cambia de manera global el aprendizaje. Generalmente se utiliza para darle un identificador único a cada iteración del aprendizaje. Ver figura 45.

4. Acciones elegidas para generar el comportamiento de la iteración.

Este campo es el más importante del reporte generado. Las acciones seleccionadas en cada iteración de *BADRL* generan un comportamiento de *n* acciones definidas en *actions.xml*. Esas *n* acciones por comportamiento son las que se observan en este campo, en el mismo orden en que fueron seleccionadas por *BADRL*. Este punto es primordial, debido a que puede ayudar a regenerar el archivo que fue generado, pues el programador puede escribir en el mismo orden las acciones que generó *BADRL* con las mismas transiciones por acción definidas en *actions.xml*. Ver figura 45.

5. La recompensa dada por BADRL para dicho comportamiento.

La recompensa es el número arrojado por la función de evaluación después de correr el comportamiento definido anteriormente mediante el sistema del *XML Behaviour Control.* La evaluación se realiza con las reglas definidas por el archivo *rules.xml* y ahí se definen cuando

mandar a llamar las recompensas definidas en *conf.xml*. Los números de ese archivo son los que salen como recompensa en el reporte, pues es con esa información que utiliza *BADRL* para determinar la recompensa por comportamiento. Ver figura 45.

6. **El promedio que maneja BADRL de la última acción elegida.**

El promedio es un número que nos indica el aprendizaje del comportamiento. Este número es arrojado de acuerdo a la recompensa de las acciones seleccionadas.

Si el promedio se encuentra cercano a -0.99 y se mantiene así, el algoritmo no aprenderá más, eso no implica que el comportamiento sea el mejor, sino que llega aun punto en el cual *BADRL* no puede seguir aprendiendo, por lo que es necesario cambiar los parámetros del archivo *conf.xml* y las recompensas para un mejor proceso de aprendizaje, de lo contrario *BADRL* sigue aprendiendo de manera correcta.

Lo anterior es muy importante pues si las recompensas dadas oscilan entre más del rango de -1 a 1, es evidente que el promedio llegará velozmente a -0.99 lo cual no ayudaría al algoritmo. Ver figura 45.

7. **La información de consola para depurar cada iteración.**

La información de depuración nos ayuda a determinar cuál regla del archivo *rules.xml* se disparó de manera tal que el archivo terminó de evaluarse. Para ello nos indica información valiosa como la posición del robot en la cancha y si éste fue capaz de controlar la bola. Estas configuraciones se pueden cambiar en el archivo *rules.xml* dentro de la etiqueta *debug*. Todo esto con el fin de brindar una mayor información al programador y determinar si el proceso de aprendizaje es de acuerdo a las necesidades del comportamiento. Ver figura 45.

QL Iteration Number	Selected Actions	Reward	Average	Debug
0	• **cabeza_der_leve** *Q-Value= 0.018747803043321465* • **cuerpo_izq_leve** *Q-Value= 0.0*	-0.05	-1.99E-4	Rule 2 - Ball NOT Found -- Info Pos = -400.0
Finish	• **cuerpo_izq_leve** *Q-Value= 0.0* • **cuerpo_izq_leve** *Q-Value= 0.0*	0.1	-3.940399E-4	TARGET REACHED!! Position = (-400.0 , -200.0)

Figura 47. Cuadro principal del reporte generado por BADRL.

Es importante señalar que durante el desarrollo de este capítulo, se mencionarán ciertas abreviaturas cuyo significado se encuentra en la Tabla 2:

Tabla 2. Abreviaturas para la evaluación de comportamientos.

BN	Behaviour Name
PA	Possible Actions
MW RW	Moving Well Reward
C RW	Collides Reward
D RW	Default Reward
R	Number of Robots
GT	Good Tests
TT	Total of Tests
FDA	Final Different Actions

8. Especificaciones técnicas del ambiente de pruebas.

A continuación se detallan los requerimientos mínimos con los cuales se realizaron las pruebas de este capítulo.

Hardware:

- *Procesador Central:*

Intel® Pentium® 4 Mobile CPU 1.60 GHz.

- *Memoria de acceso aleatorio (RAM):*

256MB

- *Disco Duro:*

20GB con espacio libre de 4GB.

Software:

- *Sistema Operativo:*
  Microsoft Windows XP Profesional. Versión 2002. Service Pack 2
- *Ambiente gráfico de programación e implementación:*
  Eclipse SDK. Versión: 3.3.0. Build id: I20070625-1500
  Visite: http://www.eclipse.org/platform
- *Lenguaje de programación (SDK & SRK):*
  Java(TM) 2 SDK, Standard Edition. Version 1.3.1
- Simulador XML Behaviour Agent Definition. Descargable en
  http://www.cem.itesm.mx/robocup

2. EVALUACIÓN DEL COMPORTAMIENTO “CONDUCCIÓN”.

Antes entrar de lleno a la evaluación de este comportamiento, será necesario definir qué es la conducción. La conducción dentro un juego de futbol se denomina a la actividad que realiza un jugador mientras tiene el control del balón, esto es, el jugador tiene la bola a una distancia en la cual es capaz de controlarla con su cuerpo y no permite que otro jugador tenga posesión de ésta

durante un determinado periodo, todo esto mientras va caminando o corriendo hacia un punto definido por el mismo jugador.

“La conducción del balón la realiza el jugador cuando domina y desplaza el balón a ras del suelo mediante una sucesión de toques con cualquier parte del pie. Es la acción técnica más natural, ya que es la que más relación guarda con la marcha, el trote y la carrera del hombre. Si bien disminuye la velocidad, es más lenta si se compara con el pase, hay situaciones en las que es fundamental, y de ella va a depender que se culmine la jugada; un jugador que se encara a la portería contraria, si lleva una buena conducción, podrá preparar el tiro, el regate, o cualquier acción para conseguir el gol.” [13]

Dentro del campo de la robótica, la conducción del balón se puede definir como el control que posee el robot sobre la bola, mientras la detecta a cierta distancia por medio de diferentes sensores: cámara, infrarrojo, de tacto, etc. A pesar de la detección, el robot debe caminar junto con la bola hacia un punto. Para ello, utiliza motores para poder tener el control de la bola. La figura 48 nos muestra un ejemplo de control de la bola de un robot de un punto A a un punto B.

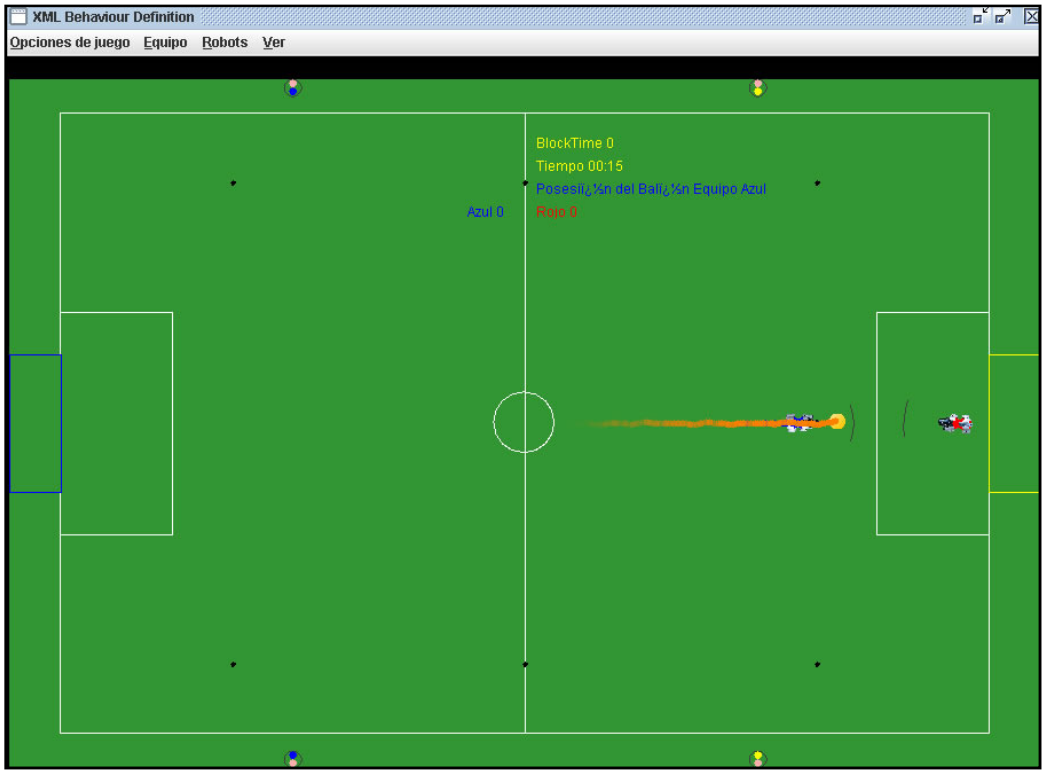


Figura 48. Conducción del balón por parte del agente de un punto A a un punto B.

BADRL, programa que genera comportamientos por medio de Q-Learning, fue utilizado para crear el comportamiento de la conducción. La regla final o meta con la cual se determina si un jugador aprendió a realizar esta actividad esta compuesta de los siguientes eventos:

1) Si el robot es capaz de ver la bola a una distancia en la cual la puede controlar, esto es, si puede realizar una patada y cambiar el estado de la bola.

2) Si el robot ve la pelota y no hay ningún obstáculo que haya entre estos dos objetos.
3) Si el robot se encuentra dentro de una posición estratégica del campo después de un determinado tiempo definido por el usuario.

Con lo anterior podemos definir si un jugador llega a la meta final del aprendizaje sobre la conducción, sin embargo es necesario indicarle algunas reglas que le indicarán si su proceso va de manera correcta o incorrecta. Para ello se utilizaron las siguientes reglas:

A. Si el agente se encuentra a una posición relativamente mayor a la que se encontraba y puede ver la bola, entonces obtiene una recompensa de MWRW.
B. Si el agente se encuentra en una posición relativamente mayor a la que se encontraba solamente, entonces obtiene una recompensa de MWRW.
C. Si el agente no avanza pero ve la bola, entonces recibe una recompensa de CRW.
D. Si el agente no avanza y no ve la bola entonces recibe una recompensa de CRW.
E. Si no ve la bola y se acabó el tiempo recibe una recompensa de CRW.
F. Si se acabó el tiempo recibe una recompensa de CRW.
G. En cualquier otro caso recibe DRW.

Por otra parte, es necesario señalar que los parámetros utilizados para la realización del comportamiento de la conducción se sacaron en base a prueba y error. Los parámetros finales se pueden observar en la Tabla 3.

Tabla 3. Parámetros definidos para el comportamiento conducción.

ALFA - param	BETA - param	GAMMA - param	Boltzmann param	TAU – param
0.5	1	0.8	Verdadero	1

Ahora bien, para poder empezar con el proceso de aprendizaje, es necesario definir las acciones que puede realizar el agente, para ello se dieron de alta los siguientes:

1. Camina lento que se dispara con la siguiente transición: Cuando ve la bola a una distancia muy cerca de su cuerpo.
2. Camina rápido que se dispara con la siguiente transición: Cuando ve la bola a una distancia cerca del cuerpo.
3. Tira a la izquierda despacio que se dispara con las transiciones:
    a. Cuando la bola puede ser pateada por el robot.
    b. Cuando la bola se encuentra a la derecha del robot.
4. Tira a la derecha despacio que se dispara con las transiciones:
    a. Cuando la bola puede ser pateada por el robot.
    b. Cuando la bola se encuentra a la izquierda del robot.
5. Tira de frente despacio que se dispara con las transiciones:
    a. Cuando la bola puede ser pateada por el robot.
    b. Cuando la bola no se encuentra ni a la izquierda ni a la derecha.
6. Tira de frente fuerte que se dispara con las transiciones:

a. Cuando la bola puede ser pateada por el robot.
b. Cuando la bola no se encuentra ni a la izquierda ni a la derecha.

El último paso para realizar un sistema de aprendizaje mediante *BADRL* es la definición del ambiente, para ello se definió lo siguiente. Para que un agente sea capaz de controlar una bola, fue necesario colocar a dicho agente frente a la bola, de manera tal que no interfiriera con el comportamiento de alinearse y aproximarse hacia ella. Con esto sólo evaluamos la conducción como tal, ahora bien, fue necesario colocar otro agente que lograra interceptar o intervenir en el comportamiento del agente y fue por ello que se colocó un portero puesto que tiene una relación lejana pero capaz de poder interceptar la bola si el agente se acerca al área de la portería.

El resultado final que arrojó *BADRL* nos indica varios detalles. El primero de ellos radica principalmente en la cantidad de pruebas que se realizaron para llegar al resultado final, pues fue el primer comportamiento que se tuvo para crear *BADRL*. Se realizaron un total de 25 corridas de las cuales al final 15 fueron contadas como correctas. En este caso se denomina una corrida correcta al proceso de aprendizaje realizado por *BADRL* que termina aprendiendo el comportamiento mediante una carrera completa. Una carrera completa se define como el número *"n"* de iteraciones de las cuales se retroalimenta el algoritmo de Q-Learning. El segundo detalle nos arroja que el comportamiento final evaluado como meta u objetivo cumplido de *BADRL* generalmente o en promedio, de las 25 corridas realizadas y de establecer que ser requería un comportamiento con 4 acciones, el comportamiento final tenía de esas 4 acciones, tan sólo 2 o 3 diferentes y aún así podía lograr el comportamiento requerido. Un ejemplo de esto se puede ver en los siguientes comportamientos en las figuras 49 y 50.

```
- <state name="firstState" iterate="true">
      <action id="MOTION-doNothing" />
    - <transition state="tira_fuerte_frente">
          <event name="VISION-shootDistance" />
          <event name="VISION-ballAtRight" operator="not" />
          <event name="VISION-ballAtLeft" operator="not" />
      </transition>
    - <transition state="camina_rapido">
          <event name="VISION-ballNear" />
          <event name="VISION-shootDistance" operator="not" />
      </transition>
    - <transition state="tira_fuerte_frente">
          <event name="VISION-shootDistance" />
          <event name="VISION-ballAtRight" operator="not" />
          <event name="VISION-ballAtLeft" operator="not" />
      </transition>
    - <transition state="camina_rapido">
          <event name="VISION-ballNear" />
          <event name="VISION-shootDistance" operator="not" />
      </transition>
  </state>
```

Figura 49. Comportamiento de conducción aprendido por *BADRL* con sólo 2 transiciones a estados diferentes.

```
- <state name="espera_inicial" iterate="true">
      <action id="MOTION-doNothing" />
   - <transition state="tira_derecha_leve">
         <event name="VISION-shootDistance" />
         <event name="VISION-ballAtLeft" />
      </transition>
   - <transition state="tira_izquierda_leve">
         <event name="VISION-shootDistance" />
         <event name="VISION-ballAtRight" />
      </transition>
   - <transition state="tira_leve_frente">
         <event name="VISION-shootDistance" />
         <event name="VISION-ballAtLeft" operator="not" />
         <event name="VISION-ballAtRight" operator="not" />
      </transition>
   - <transition state="camina_lento">
         <event name="VISION-ballVeryNear" />
         <event name="VISION-shootDistance" operator="not" />
      </transition>
  </state>
```

Figura 50. Comportamiento de conducción creado manualmente con 4 transiciones a estados diferentes

3. EVALUACIÓN DEL COMPORTAMIENTO "BÚSQUEDA DE LA BOLA".

Antes entrar de lleno a la evaluación de este comportamiento, será necesario definir qué es la búsqueda de la bola. La búsqueda de la bola dentro un juego de futbol se denomina a la actividad que realiza un jugador mientras ubica la posición de la bola dentro del terreno de juego. Esta actividad generalmente se realiza de manera natural, ya que la visión de todos los jugadores permite visualizar todo el campo. Sin embargo es importante señalar que en ocasiones el jugador debido a la posición en la que se encuentra y los jugadores en movimiento surten cierto efecto. Este efecto logra que por momentos un jugador no pueda ver de manera directa el balón, sin embargo tiene una cierta noción del área donde posiblemente se ubica el balón. Por otra parte cada jugador tiene la ayuda de sus compañeros y esto se puede observar cuando un jugador no ubica la bola, lo comunica a algún jugador cercano y éste le indica el área donde se encuentra el balón.

Dentro del campo de la robótica, la ubicación del balón dentro del terreno de juego es un tema de especial interés. Lo anterior se define debido a las capacidades tecnológicas que actualmente poseen las cámaras de video utilizadas por los robots. En el caso concreto de la cámara de los Sony AIBO [1], ésta es deficiente en varios aspectos:

- El robot tiene una cámara CCD en la cabeza. Se pueden obtener hasta 25 imágenes YUV por segundo de esa cámara [Figura 46].

- Las opciones de resolución que tiene son 3:
    - Alta: 172 * 144 pixeles.
    - Media: 88 * 72 pixeles.
    - Baja: 44 * 36 pixeles.
- Las opciones del modo son:
    - Indoor.
    - Outdoor.
    - Flouroscent
- Las opciones de ganancia son:
    - Alta.
    - Media.
    - Baja.
- Las opciones de la velocidad de captura son:
    - Rápida.
    - Media.
    - Lenta.

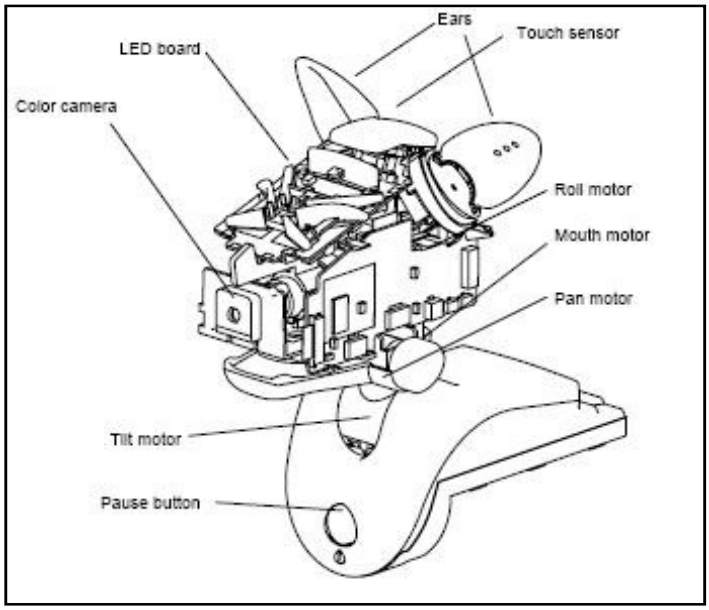


Figura 51. Tamaño y posición de la cámara de color localizada dentro del robot AIBO modelo 210.

Dado lo anterior, se puede determinar que la búsqueda de la bola no es tan natural como sería como en los humanos. Es por ello que se debe determinar diferentes maneras para que la cámara de los robots detecte la bola.

Para el caso particular de esta tesis se utilizó el *XML Behaviour Control* [16] como interfaz para ser utilizada por *BADRL* y así generar el comportamiento de la búsqueda de la bola. La regla final o meta con la cual se determina si un jugador aprendió a realizar esta actividad se evalúa si el robot es capaz de ver la bola por medio de la activación evento *VISION-ballFound*.

Con lo anterior podemos definir si un jugador llega a la meta final del aprendizaje sobre la búsqueda de la bola, sin embargo es necesario indicarle algunas reglas que le indicarán si su proceso va de manera correcta o incorrecta. Para ello sólo se utilizó la siguiente regla: Si el evento VISION-ballFound se encuentra inactivo, por lo tanto recibe una recompensa de CRW, cualquier otro caso se recompensa con DRW.

Por otra parte, es necesario señalar que los parámetros utilizados para la realización del comportamiento de la búsqueda de la bola se sacaron en base a prueba y error. Y los parámetros finales se pueden observar en la Tabla 4.

Tabla 4. Parámetros definidos para el comportamiento búsqueda de la bola.

ALFA - param	BETA - param	GAMMA - param	Boltzmann param	TAU – param
0.5	1	0.8	Verdadero	1

Ahora bien, para poder empezar con el proceso de aprendizaje, es necesario definir las acciones que puede realizar el agente, para ello se dieron de alta los siguientes:

1. Mueve el cuerpo hacia la izquierda de manera lenta que se dispara con la siguiente transición: Cuando ve la bola a la izquierda del robot.
2. Mueve el cuerpo hacia la derecha de manera lenta que se dispara con la siguiente transición: Cuando ve la bola a la derecha.
3. Mueve la cabeza a la izquierda de manera lenta que se dispara con la siguiente transición: Cuando ve la bola a la izquierda del robot.
4. Mueve la cabeza a la derecha despacio que se dispara con la siguiente transición: Cuando ve la bola a la derecha del robot.

En este punto existe una anotación que se debe apuntar. ¿Cómo es posible evaluar si un agente logró encontrar la bola si las acciones se disparan con transiciones que dependen de la posición de la bola? Esta pregunta es interesante en el sentido que suena un tanto recíproco. La respuesta de esta pregunta se puede efectuar en 2 puntos primordialmente.

El primero de ellos se ubica dentro de la interfaz *XML Behaviour Control* [16]. Este simulador permite que el agente siempre vea la bola, pues la posición de ésta es guardada dentro de la interfaz primordial que controla todas las acciones que suceden dentro del ambiente de simulación. Por lo tanto si un agente es programado para moverse a encontrar la bola lo hará pues siempre sabe dónde está. Para ello, fue necesario utilizar las acciones con las transiciones de esa manera.

El segundo de ellos se ubica en la definición de lo que significa la transición “Cuando ve la bola ya sea a la izquierda o derecha del robot”. Esto no significa que el robot vea la bola, sino que el simulador le indica al agente si la bola está a su izquierda o derecha; una cosa es que el agente vea el balón y por otra que se le indique que la bola está a su izquierda o derecha. Evidentemente

esta anotación resulta de gran ayuda, sin embargo se puede tener una correlación con la manera natural de búsqueda de la bola con el humano, pues a pesar de que en muy pocas ocasiones pierde la posición exacta de la bola, al menos conoce el área posible o los mismos compañeros le indican por dónde y es en este punto donde se puede contrarrestar esta ayuda en el campo de la simulación, pues el simulador se puede ver como un compañero que le indica por donde más o menos se encuentra la bola respecto a su posición en el campo y es por ello que se utilizaron estas transiciones [Figura 52].

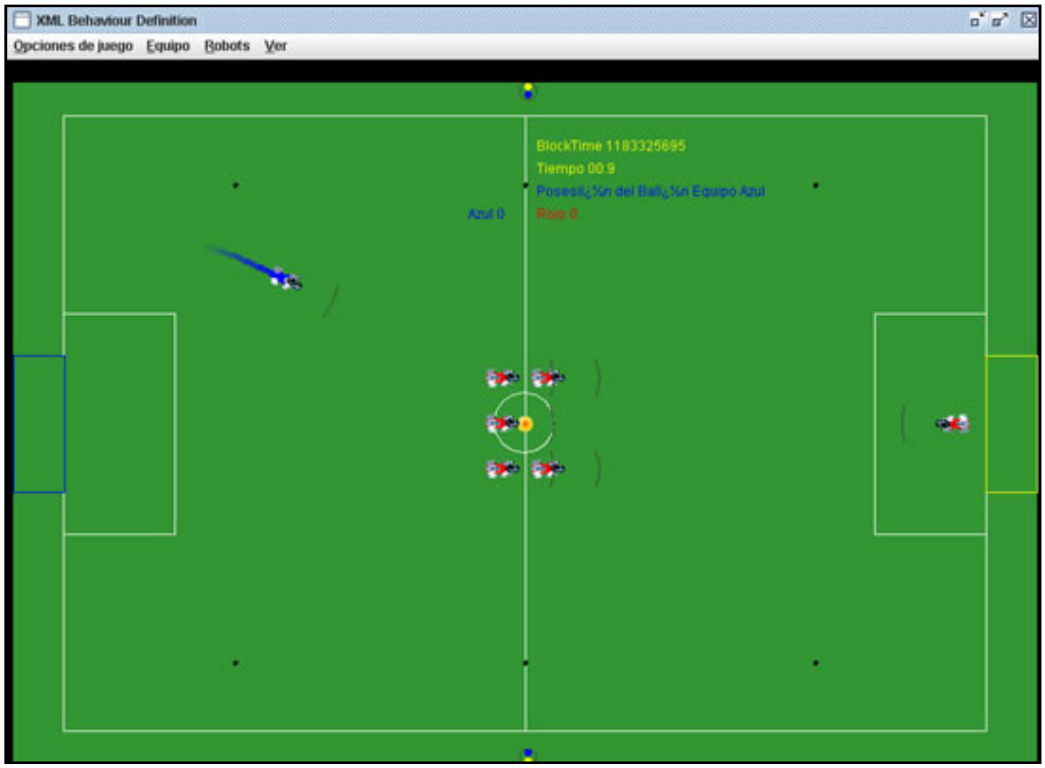


Figura 52. Alineación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control*

El último paso para realizar un sistema de aprendizaje mediante *BADRL* es la definición del ambiente, para ello se definió lo siguiente. Para que un agente sea capaz de buscar la bola o alinearse al menos ante ella, se definió un ambiente en el cual la bola se encuentre en medio de 6 jugadores con un comportamiento simple como el do-Nothing. Lo anterior permite que el agente que tiene que aprender este comportamiento no lo tenga tan visiblemente de frente, además el agente se colocó de inicio observando al extremo opuesto a donde se encuentra la bola.

El resultado final que arrojó *BADRL* nos indica varios detalles. El primero de ellos radica principalmente en la cantidad de pruebas que se realizaron para llegar al resultado final. Se realizaron un total de 15 corridas de las cuales las 15 fueron contadas como correctas.

En este caso se denomina una corrida correcta al proceso de aprendizaje realizado por *BADRL* que termina aprendiendo el comportamiento mediante una carrera completa. Una carrera

completa se define como el número *"n"* de iteraciones de las cuales se retroalimenta el algoritmo de Q-Learning.

El segundo detalle nos arroja que el comportamiento final evaluado como meta u objetivo cumplido de *BADRL* generalmente o en promedio, de las 25 corridas realizadas y de establecer que se requería un comportamiento con 2 acciones, el comportamiento final tenía de esas 2 acciones, tan sólo 1 o 2 diferentes y aún así podía lograr el comportamiento requerido. Un ejemplo de esto se puede ver en los siguientes comportamientos en las figuras 53 y 54.

```
- <state name="firstState" iterate="true">
      <action id="MOTION-doNothing" />
    - <transition state="cuerpo_izq_leve">
        <event name="VISION-ballAtLeft" />
      </transition>
    - <transition state="cuerpo_izq_leve">
        <event name="VISION-ballAtLeft" />
      </transition>
  </state>
```

Figura 53. Comportamiento de búsqueda la bola aprendido por *BADRL* con sólo 1 transición a estado diferente.

```
- <state name="firstState" iterate="true">
    <action id="MOTION-doNothing" />
    <transition state="voltea_cuerpo_izquierda_leve">
      <event name="VISION-ballAtLeft" />
    </transition>
  - <transition state="voltea_cuerpo_derecha_leve">
      <event name="VISION-ballAtRight" />
    </transition>
  - <transition state="voltea_cabeza_izquierda_leve">
      <event name="VISION-ballAtLeft" />
    </transition>
  - <transition state="voltea_cabeza_derecha_leve">
      <event name="VISION-ballAtRight" />
    </transition>
  </state>
```

Figura 54. Comportamiento de búsqueda de la bola creado manualmente con 4 transiciones a estados diferentes.

4. EVALUACIÓN DEL COMPORTAMIENTO "APROXIMACIÓN DE LA BOLA".

Antes entrar de lleno a la evaluación de este comportamiento, será necesario definir qué es la aproximación de la bola. La aproximación de la bola dentro un juego de futbol se denomina a la actividad que realiza un jugador cuando ve la bola en su área de juego y se acerca a ella para

poder estar con el balón tan cerca de manera tal que pueda manipularla. Es importante señalar que debido a que el juego del fútbol es sumamente dinámico, por lo tanto un jugador puede perder el esférico dentro de la trayectoria que tenía principalmente trazada para llegar al esférico. Se entiende como dinámico a la capacidad que tienen todos los jugadores de ir tras el balón y cambiar el ambiente en el que se encuentran de manera rápida. Cada jugador de acuerdo a cierta estrategia definida anteriormente por un director técnico, el jugador es capaz de ir por la bola de acuerdo a la posición que éste tenga respecto a ella.

Dentro del campo de la robótica, la aproximación de la bola es un tema muy especial, pues este tema depende de la dinámica del juego. Para un robot es relativamente fácil trazar un camino en línea recta para llegar de un punto A a un punto B, sin embargo si ese punto B cambia continuamente, el robot tratará de llegar a cada uno de esos puntos por lo que tardará un cierto tiempo *t* en tratar de llegar al último punto B en el cual la bola se encuentra totalmente parada. Es importante enseñarle al robot que sea capaz de determinar cuál será ese último punto B para que su movimiento vaya directamente a éste si el robot se encuentra lejano a la trayectoria de la bola. Por el contrario, si el robot se encuentra relativamente cerca de la trayectoria de la bola y puede recuperarla, por lo tanto puede interceptar en algún punto C el balón.

Por otra parte, es necesario señalar que los parámetros utilizados para la realización del comportamiento de la búsqueda de la bola se sacaron en base a prueba y error. Y los parámetros finales se pueden observar en la Tabla 5.

Tabla 5. Parámetros definidos para el comportamiento aproximación de la bola.

ALFA – param	BETA – param	GAMMA – param	Boltzmann param	TAU – param
0.5	1	0.8	Verdadero	1

El último paso para realizar un sistema de aprendizaje mediante *BADRL* es la definición del ambiente, para ello se definió lo siguiente. Para que un agente sea capaz de aproximarse a la bola, se definió un ambiente en el cual la bola se encuentre en medio de 6 jugadores con un comportamiento simple como el do-Nothing. Lo anterior permite que el agente que tiene que aprender este comportamiento no lo tenga tan visiblemente de frente, además el agente se colocó de inicio observando de frente a la bola. [Figuras 55 y 56]

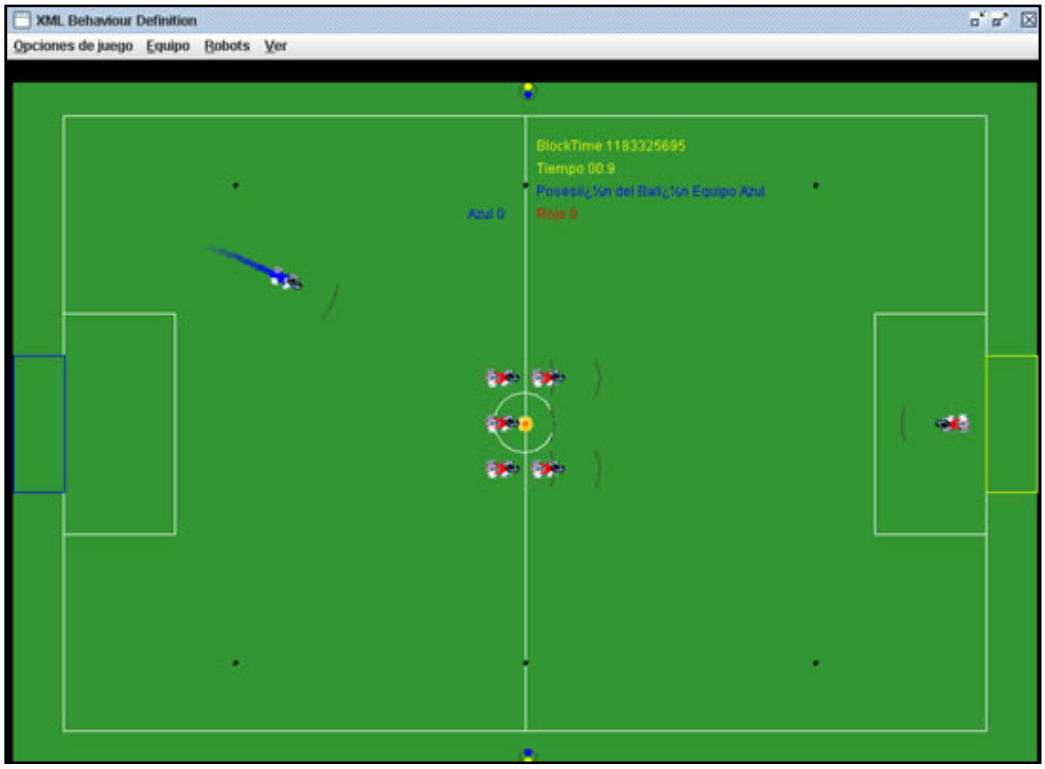


Figura 55. Aproximación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control.*

```
- <state name="firstState" iterate="true">
    <action id="MOTION-doNothing" />
  - <!--
        APPROACH BALL Tarda 1:15 minutos
    -->
  - <transition state="voltea_cuerpo_derecha_leve">
        <event name="VISION-ballFound" operator="not" />
    </transition>
  - <transition state="camina_frente_leve">
        <event name="VISION-ballFound" />
        <analogEvent name="angleToBall" operator="greaterOrEqual" value="350" />
    </transition>
  - <transition state="camina_frente_leve">
        <event name="VISION-ballFound" />
        <analogEvent name="angleToBall" operator="lessOrEqual" value="10" />
    </transition>
  - <transition state="voltea_cuerpo_izquierda_leve">
        <event name="VISION-ballFound" />
        <event name="VISION-ballAtLeft" />
    </transition>
  - <transition state="voltea_cuerpo_derecha_leve">
        <event name="VISION-ballFound" />
        <event name="VISION-ballAtRight" />
    </transition>
  - <transition state="voltea_cabeza_izquierda_leve">
        <event name="VISION-ballFound" />
```

```
      <event name="VISION-ballAtLeft" />
    </transition>
  - <transition state="voltea_cabeza_derecha_leve">
      <event name="VISION-ballFound" />
      <event name="VISION-ballAtRight" />
    </transition>
  </state>
```

Figura 56. Comportamiento de acercarse a la bola creado manualmente con 7 transiciones a estados diferentes

5. EVALUACIÓN DEL COMPORTAMIENTO "DISPARO A PORTERÍA".

Antes entrar de lleno a la evaluación de este comportamiento, será necesario definir qué es la disparo a la portería. El disparo a la portería dentro un juego de futbol se denomina a la actividad que realiza un jugador cuando el agente ve la bola cerca de éste y es capaz de ejecutar un tiro directo a la portería. Es importante señalar que debido a que el juego del fútbol es sumamente dinámico, por lo tanto un jugador puede perder el esférico dentro de la trayectoria que tenía principalmente trazada para llegar al esférico. Sin embargo, este comportamiento sólo se dedica a disparar a la portería sin importar cuantos factores externos puedan modificar el disparo.

Dentro del campo de la robótica, la aproximación de la bola es un tema muy especial, pues este tema depende de la dinámica del juego. Para un robot es relativamente fácil disparar en línea recta un objeto como lo es el balón.

Es importante enseñarle al robot que sea capaz de determinar cuál será la mejor manera de disparar a portería de acuerdo a los factores externos que existen en el ambiente, pues puede ser que el portero decida lanzarse a la derecha o a la izquierda, o decida salirse o decida mandar a llamar a un defensa que bloquee el disparo. Como podemos ver, esta gama de posibilidades tiene que ser analizada por nuestro agente y tiene que aprender a realizar la acción de la mejor manera posible, pues su objetivo final es la anotación en esa portería.

Por otra parte, es necesario señalar que los parámetros utilizados para la realización del comportamiento del disparó a portería se sacaron en base a prueba y error. Y los parámetros finales se pueden observar en la Tabla 6.

Tabla 6. Parámetros definidos para el comportamiento disparo a portería.

ALFA – param	BETA - param	GAMMA – param	Boltzmann param	TAU – param
0.5	1	0.8	Verdadero	1

El último paso para realizar un sistema de aprendizaje mediante *BADRL* es la definición del ambiente, para ello se definió lo siguiente. Para que un agente sea capaz de disparar a portería, se definió un ambiente en el cual la bola se encuentre afuera del área del área del portero y éste agarre velocidad para ejecutar un disparo, ya sea de frente, a la derecha o la izquierda del portero a quema ropa. [Figuras 57 y 58].

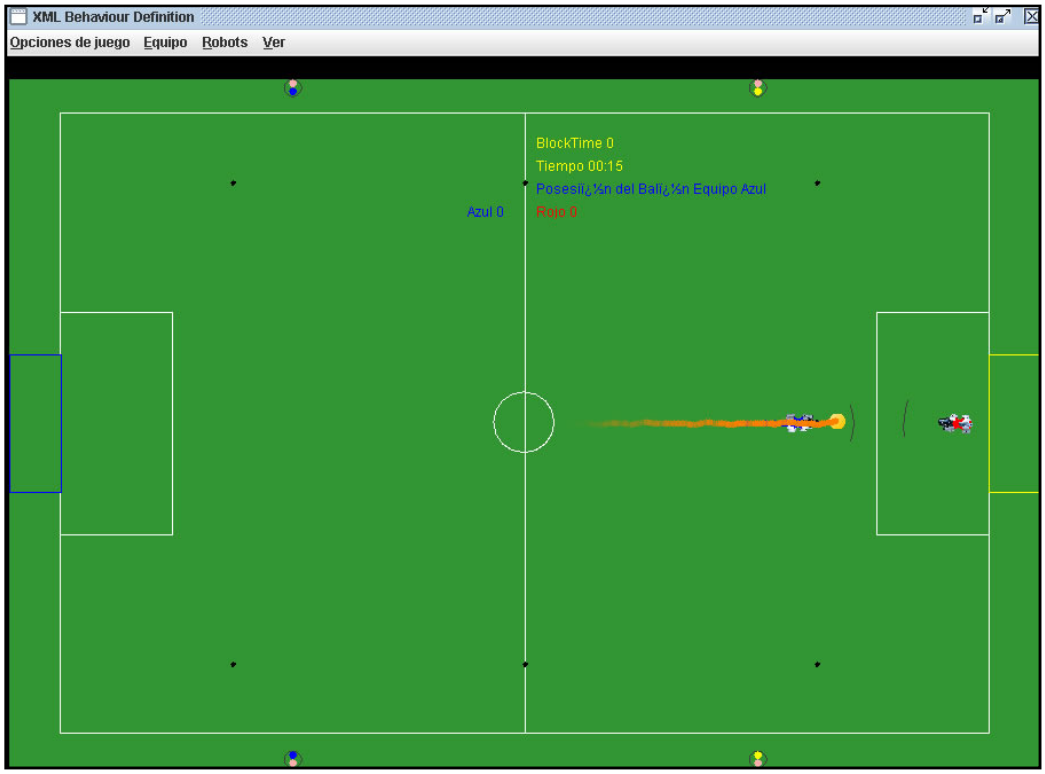


Figura 57. Aproximación del agente respecto a la bola dentro del ambiente de pruebas del *XML Behaviour Control.*

```
<!--  DEBUG XMLWriting: BADRL STATES  -->
- <state name="firstState" iterate="true">
    <action id="MOTION-doNothing" />
  - <transition state="camina_frente_medio">
      <event name="VISION-ballVeryNear" />
      <event name="VISION-shootDistance" operator="not" />
    </transition>
  - <transition state="tira_derecha_fuerte">
      <event name="VISION-shootDistance" />
    </transition>
  </state>
```

Figura 58. Comportamiento de disparar a la portería creado por BADRL con 2 transiciones a estados diferentes.

6. EVALUACIÓN DEL COMPORTAMIENTO "TRIANGULACIÓN PARA DISPARO A PORTERÍA".

Antes entrar de lleno a la evaluación de este comportamiento, será necesario definir qué es la triangulación para disparo a portería. Una triangulación está definida como la acción que realiza un jugador *J1* de pasar el esférico a un jugador *J2* y qué éste la regrese al primero *(J1)* cuando el primer jugador *(J1)* haya cambiado de posición. Esta jugada se conoce generalmente como pared pues el jugador J2 funciona como ello. Por otro lado es importante señalar que existe una triangulación en el momento mismo que el jugador J2 al momento de recibir y regresar el esférico de dos puntos distintos provoca que la jugada desde un vista superior se vea como un triangulo.

Dentro del campo de la robótica, la triangulación representa un reto especial, pues en general deben existir al menos 2 agentes para la realización de este comportamiento. En el momento mismo que se sugiere utilizar más de un agente, es aquí donde entra la complejidad para la generación de comportamientos entre agentes colaborativos. Pues si bien un agente J1 mandará y recibirá el esférico, el agente J2 hará lo contrapuesto con la diferencia única de que no cambiará su posición inicial como el jugador J1.

Tabla 7. Parámetros definidos para el comportamiento triangulación para disparo a portería.

ALFA – param	BETA – param	GAMMA – param	Boltzmann param	TAU – param
0.5	1	0.8	Verdadero	1

El último paso para realizar un sistema de aprendizaje mediante *BADRL* es la definición del ambiente, para ello se definió lo siguiente. Para que un agente sea capaz de triangular y disparar a portería, se definió un ambiente en el cual la bola se encuentre en el centro del medio y el jugador J1 mande el esférico a un jugador J2 con un comportamiento manual de recibir la bola y mandarla a un punto dado, en lo que el jugador J1 aprende a llegar al punto para recibir la bola y después disparar a portería [Figura 59 y 60].

```
- <state name="firstState" iterate="true">
    <complexAction name="approachingWithAligningsBADRL" />
  - <transition state="tira_izquierda_porteria">
      <event name="VISION-shootDistance" />
    </transition>
  - <transition state="tira_izquierda_porteria">
      <event name="VISION-shootDistance" />
    </transition>
  - <transition state="tira_izquierda_porteria">
      <event name="VISION-shootDistance" />
    </transition>
  </state>
```

Figura 59. Comportamiento de mandar, recibir la bola y disparar a la portería creado por BADRL con 4 transiciones a estados diferentes.

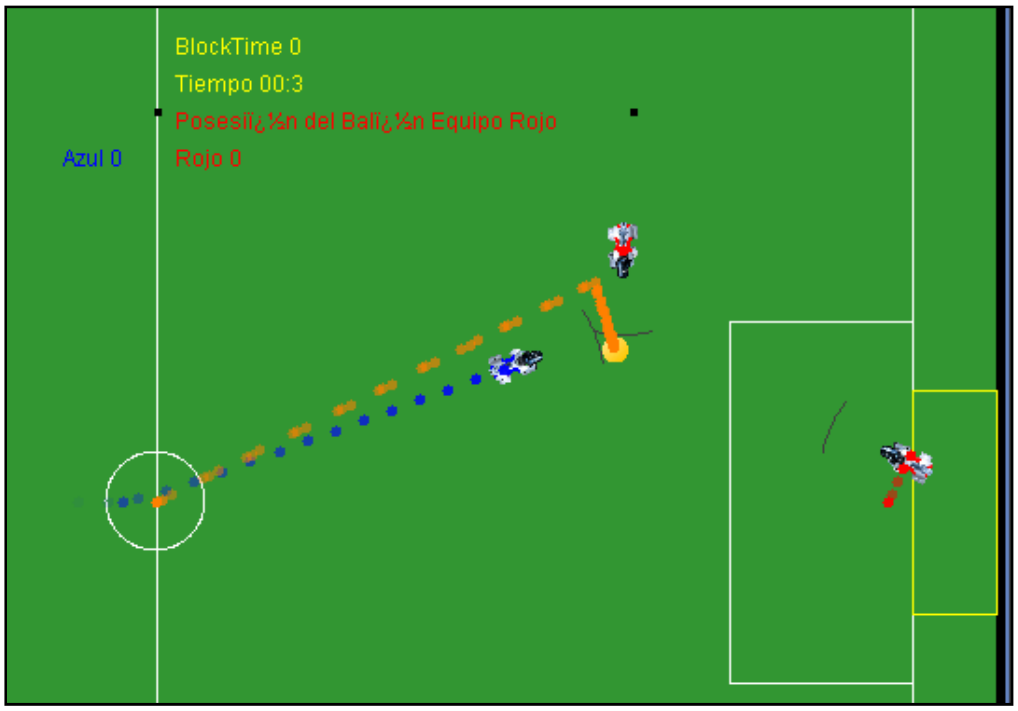


Figura 60. Ambiente de pruebas para la triangulación antes del disparo a portería.

7. ANÁLISIS Y CONCLUSIONES DE LOS COMPORTAMIENTOS GENERADOS POR *BADRL*.

A. RESULTADOS PARA EL COMPORTAMIENTO BÚSQUEDA DE LA BOLA.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones.

De 15 pruebas realizadas, BADRL siempre llegó a la respuesta esperada en un tiempo promedio de 14.06 segundos. Además utilizó 1.5 iteraciones con 4 acciones posibles por utilizar.

De manera general, BADRL de las 4 acciones posibles sólo utilizo 1.53 acciones, de las cuales la que más se utilizó fue mover el cuerpo hacia la izquierda, seguido de mover el cuerpo a la derecha con un promedio de veces de 0.53.

B. RESULTADOS PARA EL COMPORTAMIENTO CONDUCCIÓN.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones sobre el comportamiento de conducción.

Tabla 8. Resultados de BADRL para el comportamiento de alineación.

Aligning								
t (s)	Iteraciones	Acciones	Target?	A1	A2	A3	A4	Total
8	1	4	SI	1	0	0	0	1
9	1	4	SI	1	0	0	0	1
9	1	4	SI	1	1	0	0	2
10	1	4	SI	1	1	0	0	2
10	1	4	SI	0	1	0	0	1
10	1	4	SI	1	1	0	0	2
10	1	4	SI	1	1	0	0	2
10	1	4	SI	1	1	1	1	4
11	1	4	SI	1	0	0	0	1
11	1	4	SI	1	1	0	0	2
17	2	4	SI	0	1	0	0	1
18	2	4	SI	1	0	0	0	1
19	2	4	SI	1	0	0	0	1
25	3	4	SI	1	0	0	0	1
34	4	4	SI	1	0	0	0	1
14.07	1.53	4		86.67%	53.33%	6.67%	6.67%	1.53333

Tabla 9. Relación de número y nombres de acciones de la tabla 8.

Acción	Nombre
A1	cuerpo_izq_leve
A2	cuerpo_der_leve
A3	cabeza_izq_leve
A4	cabeza_der_leve

Tabla 10. Resultados de BADRL para el comportamiento de conducción.

Dribbling												
t (s)	Iterac.	Accion.	Target	A1	A2	A3	A4	A5	A6	A7	A8	Total
18	2	8	NO	0	0	0	1	0	1	0	0	2
68	7	8	SI	0	1	0	0	1	0	1	0	3
112	6	6	SI	0	1	0	0	1	0	0	1	3
142	7	8	NO	0	0	0	0	1	1	1	0	3
153	10	6	SI	0	1	0	0	0	0	1	0	2
210	24	8	SI	1	0	0	0	1	1	0	1	4
215	32	4	SI	1	0	0	0	1	0	0	1	3
234	33	4	NO	1	0	0	0	1	0	0	0	2
252	36	4	SI	1	0	0	0	1	1	0	1	4
252	36	4	SI	1	0	0	0	1	1	0	1	4
254	15	8	SI	0	1	0	0	0	0	1	0	2
284	33	8	SI	0	1	0	0	1	1	1	0	4
308	36	4	SI	1	0	0	0	1	0	0	1	3
356	21	8	NO	1	0	0	0	0	1	0	0	2
416	18	4	SI	1	0	0	0	1	0	0	1	3
559	66	4	SI	1	0	0	0	1	1	0	1	4
563	57	4	SI	1	0	0	0	1	1	0	1	4
579	27	4	SI	1	0	0	0	1	0	0	1	3
593	70	8	SI	0	1	0	0	0	0	1	0	2
593	70	8	SI	0	1	0	0	0	0	1	0	2
640	234	4	NO	1	0	0	0	0	0	0	0	1
717	91	4	SI	1	0	0	0	1	0	0	1	3
882	106	4	NO	1	0	0	0	1	0	0	0	2
947	114	4	SI	1	0	0	0	0	1	0	0	2
1077	58	6	SI	1	0	0	0	1	0	0	1	3
1654	202	4	NO	1	0	0	0	1	0	0	0	2
1680	87	8	SI	1	0	0	0	0	1	0	0	2
1770	200	8	NO	0	1	0	1	1	0	0	0	3
3536	183	6	SI	1	0	0	0	1	0	0	1	3
3751	201	6	NO	0	0	0	0	1	1	0	0	2
760.5	69.4	5.8		63.3%	26.7%	0.0%	6.7%	70.0%	40.0%	23.3%	43.3%	2.7

Tabla 11. Relación de número y nombres de acciones de la tabla 10.

Acción	Nombre
A1	camina_lento
A2	camina_rapido
A3	tira_derecha_fuerte
A4	tira_izquierda_fuerte
A5	tira_izquierda_leve
A6	tira_derecha_leve
A7	tira_fuerte_frente
A8	tira_leve_frente

Tabla 12. Porcentaje de efectividad de BADRL para el comportamiento conducción.

SI	21	70.00%
NO	9	30.00%

De 30 pruebas realizadas, BADRL llegó a la respuesta esperada (target) en un 70% de las veces en un tiempo promedio de 766 segundos. Además utilizó 69.4 iteraciones con 5.8 acciones posibles por utilizar en promedio

De manera general, BADRL de las 5.8 acciones posibles sólo utilizo 2.73 acciones, de las cuales las que más se utilizó fue mover el caminar lento, seguido de tirar el balón despacio de frente con un promedio de veces de 0.433.

C. RESULTADOS PARA EL COMPORTAMIENTO DISPARO A PORTERÍA.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones.

Tabla 13. Resultados de BADRL para el comportamiento disparo a portería.

Kicking											
t (s)	Iteraciones	Acciones	Target?	A1	A2	A3	A4	A5	A6	A7	Total
7	1	3	SI	1	1	0	0	0	0	0	2
7	1	3	SI	1	1	0	0	0	0	0	2
7	1	3	SI	1	1	0	0	0	0	0	2
7	1	3	SI	1	1	0	0	0	0	0	2
7	1	3	SI	1	1	0	0	0	0	0	2
8	1	3	SI	1	0	1	0	0	0	0	2
11	1	7	SI	1	1	0	0	0	0	0	2
11	1	7	SI	1	1	0	0	0	0	0	2
14	1	4	SI	1	1	0	0	0	0	0	2
24	3	3	SI	1	1	0	0	0	0	0	2
32	3	4	SI	1	1	0	0	0	0	0	2
32	3	4	SI	1	1	0	0	0	0	0	2
32	4	3	SI	1	0	1	0	0	0	0	2
39	5	3	SI	1	0	1	0	0	0	0	2
40	5	3	SI	1	1	0	0	0	0	0	2
45	6	3	SI	1	1	0	0	0	0	0	2
49	4	4	SI	1	0	1	0	0	0	0	2
52	4	4	SI	1	1	0	0	0	0	0	2
54	4	4	SI	1	0	1	0	0	0	0	2
55	7	3	SI	1	1	0	0	0	0	0	2
61	7	3	SI	1	1	0	0	0	0	0	2
63	5	4	SI	1	1	0	0	0	0	0	2
63	8	3	SI	1	1	0	0	0	0	0	2
92	12	3	SI	1	1	0	0	0	0	0	2
104	9	4	SI	1	0	1	0	0	0	0	2
114	10	4	SI	1	1	0	0	0	0	0	2

124	11	4	SI	1	1	0	0	0	0	0	2
501	50	4	NO	1	1	0	0	0	0	0	2
1317	70	7	NO	0	0	0	1	1	0	0	2
2012	190	4	NO	1	1	0	0	0	0	0	2
166.13	14.3	3.8		96.67%	76.67%	20.00%	3.33%	3.33%	0.00%	0.00%	2

Tabla 14. Relación de número y nombres de acciones de la tabla 13.

Acción	Nombre
A1	camina_lento
A2	tira_derecha_fuerte
A3	tira_izquierda_fuerte
A4	tira_izquierda_leve
A5	tira_derecha_leve
A6	tira_fuerte_frente
A7	tira_leve_frente

Tabla 15. Porcentaje de efectividad de BADRL para el comportamiento disparo a portería.

SI	27	90.00%
NO	3	10.00%

De 30 pruebas realizadas, BADRL llegó a la respuesta esperada (target) en un 90% de las veces en un tiempo promedio de 166.1 segundos. Además utilizó 14.3 iteraciones con 3.8 acciones posibles por utilizar en promedio.

De manera general, BADRL de las 3.8 acciones posibles sólo utilizo 2 acciones, de las cuales las que más se utilizó fue mover el caminar lento un 0.96 veces, seguido de tirar el balón fuerte a la derecha del portero con un promedio de veces de 0.766.

D. RESULTADOS PARA EL COMPORTAMIENTO SEGUIR LA BOLA.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones.

De 18 pruebas realizadas, BADRL llegó a la respuesta esperada (target) en un 88.88% de las veces en un tiempo promedio de 74.2 segundos. Además utilizó 3.22 iteraciones con 3 acciones posibles por utilizar en promedio.

De manera general, BADRL de las 3 acciones posibles sólo utilizo 2 acciones, de las cuales las que más se utilizó fue mover el alinearse a la bola en un 0.944 veces, seguido de caminar rápido de frente con un promedio de veces de 0.777.

Tabla 16. Resultados de BADRL para el comportamiento seguir la bola.

Approaching & Aligning							
T (s)	Iteraciones	Acciones	Target?	A1	A2	A3	Total
20	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	1	0	2
21	1	3	SI	1	0	1	2
21	1	3	NO	0	1	1	2
62	3	3	SI	1	0	1	2
81	4	3	SI	1	1	0	2
83	3	3	SI	1	0	1	2
103	4	3	SI	1	1	0	2
104	4	3	SI	1	1	0	2
123	5	3	SI	1	1	0	2
124	5	3	SI	1	1	0	2
144	6	3	SI	1	0	1	2
325	15	3	NO	1	1	0	2
74.28	3.22	3.00		94.44%	77.78%	27.78%	2

Tabla 17. Relación de número y nombres de acciones de la tabla 16.

Acción	Nombre
A1	manual_Aligning
A2	camina_frente_mas
A3	camina_frente_menos

Tabla 18. Porcentaje de efectividad de BADRL para el comportamiento seguir la bola.

SI	16	88.88%
NO	2	11.11%

E. RESULTADOS PARA EL COMPORTAMIENTO TRIANGULACIÓN PARA DISPARO A PORTERÍA.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones.

De 24 pruebas realizadas, BADRL llegó a la respuesta esperada el 75% de las veces en un tiempo promedio de 290.6 segundos. Además utilizó 32.92 iteraciones con 4 acciones posibles por utilizar.

De manera general, BADRL de las 4 acciones posibles sólo utilizo 1.96 acciones, de las cuales la que más se utilizó fue ir por la bola el 91.6% de las veces, seguido de tirar a la izquierda al compañero y a la portería con un promedio de veces de 0.58.

Tabla 19. Resultados de BADRL para el comportamiento triangulación para disparo a portería.

Helper Passing								
t (s)	Iteraciones	Acciones	Target?	A1	A2	A3	A4	Total
16	2	4	SI	1	0	1	0	2
20	2	4	SI	1	1	0	0	2
22	3	4	SI	1	0	1	0	2
24	3	4	SI	1	0	1	0	2
30	4	4	SI	1	0	1	0	2
30	4	4	SI	1	1	0	0	2
32	4	4	SI	1	1	0	0	2
32	4	4	SI	1	1	0	0	2
36	6	4	SI	1	0	1	0	2
40	4	4	SI	1	0	1	0	2
48	5	4	SI	1	1	0	0	2
56	7	4	SI	1	0	1	0	2
57	6	4	SI	1	1	0	0	2
95	11	4	SI	1	1	0	0	2
97	11	4	SI	1	1	0	0	2
125	16	4	SI	1	0	1	0	2
139	18	4	NO	1	1	0	0	2
211	25	4	SI	1	1	0	0	2
253	18	4	NO	1	1	0	0	2
533	65	4	SI	1	1	1	0	3
1125	143	4	NO	0	0	0	1	1
1185	143	4	NO	1	0	0	1	2
1216	143	4	NO	0	1	0	0	1
1554	143	4	NO	1	1	0	0	2
290.67	32.92	4.00		91.67%	58.33%	37.50%	8.33%	1.96

Tabla 20. Relación de número y nombres de acciones de la tabla 19.

Acción	Nombre
A1	approachingWithAligningsBADRL
A2	tira_izquierda_lejos
A3	tira_izquierda_porteria
A4	tira_derecha_porteria

Tabla 21. Porcentaje de efectividad de BADRL para el comportamiento triangulación para disparo a portería.

SI	18	75.00%
NO	6	25.00%

F. RESULTADOS PARA EL COMPORTAMIENTO EVADIR OBSTÁCULOS.

Para realizar este comportamiento, BADRL fue utilizado para generar diferentes pruebas y obtener resultados consistentes que nos determinan varias conclusiones.

Tabla 22. Resultados de BADRL para el comportamiento evadir obstáculos.

Avoiding									
Tiempo (seg)	Iteraciones	Acciones	Target?	A1	A2	A3	A4	A5	T
156	16	5	NO	1	1	1	1	0	4
1125	128	5	NO	0	0	1	1	1	3
1125	128	5	NO	1	1	1	1	1	5
1125	128	5	NO	0	0	1	1	0	2
1125	128	5	NO	0	0	1	1	0	2
1125	128	5	NO	0	1	0	0	1	2
1125	128	5	NO	1	0	0	1	1	3
1125	128	5	NO	0	1	1	0	0	2
1125	128	5	NO	1	1	1	1	0	4
1125	128	5	NO	1	1	0	1	1	4
1125	128	5	NO	1	1	1	0	0	3
1125	128	5	NO	0	1	1	1	1	4
1125	128	5	NO	1	0	1	0	1	3
1050.46	119.38	5.00		53.85%	61.54%	76.92%	69.23%	53.85%	3.15

Tabla 23. Relación de número y nombres de acciones de la tabla 22.

Acción	Nombre
A1	camina_derecha_leve
A2	camina_izquierda_leve
A3	camina_frente_medio
A4	voltea_cuerpo_izq
A5	voltea_cuerpo_der

Tabla 24. Porcentaje de efectividad de BADRL para el comportamiento evadir obstáculos.

SI	0	0
NO	13	100

De 13 pruebas realizadas, BADRL no pudo llegar a la respuesta esperada en un tiempo promedio de 1125 segundos. Además utilizó 128 iteraciones con 5 acciones posibles por utilizar.

De manera general, BADRL de las 5 acciones posibles utilizó 3.15 acciones, de las cuales la que más se utilizó fue caminar de frente rápido con un 77% de las veces, voltear el cuerpo a la izquierda con un 69% y caminar a la izquierda despacio en un 61% de las veces.

Este comportamiento es especial, debido a que BADRL no pudo llegar a la solución del problema, a pesar de que explotó la memoria de la computadora y utilizar en promedio la máxima vista de iteraciones. Como se puede dar cuenta en la tabla, BADRL utilizó casi en la misma proporción cada acción por comportamiento, lo que nos indica claramente que estuvo la herramienta estuvo tratando por todos los medios en llegar a la meta (*target*) y no pudo. Este comportamiento es un ejemplo que resulta difícil para la herramienta encontrar esta respuesta usando Q-Learning, además otro factor que influye son las acciones, pues pudiera ser que faltaron y sobraron acciones fundamentales para el aprendizaje de este comportamiento.

Lo anterior representa un punto importante en la investigación, este es, por qué el algoritmo Q-Learning no es capaz de llegar a lograr y generar comportamientos específicos. En primer lugar cabe señalar que Q-Learning al ser un algoritmo de diferencia temporal, va cambiando sus estados internos de acuerdo a las recompensas que se generar por medio de la evaluación de su medio. Si el medio es constante y no genera cambios representativos para el agente, éste tomará las mismas recompensas pues para el agente durante su evolución del aprendizaje, el medio fue exactamente el mismo. Por otro lado, en el caso específico del comportamiento de obstrucción de obstáculos, el agente simulado tiene acciones muy limitadas que no le permiten lograr ese comportamiento. De hecho, se trató de hacer manera manual, sin embargo tampoco resultó exitoso.

Esto último no representa que BADRL sea incapaz de llegar a una solución para ese comportamiento, pues de acuerdo a la complejidad del comportamiento de evadir obstáculos, era necesariamente generar sub comportamientos para generar el comportamiento final complejo, sin embargo debido al tiempo en que se desarrolló esta tesis, no fue posible generar estas pruebas. En conclusión, Q-Learning depende de las acciones que tiene el agente, si el agente no tiene la capacidad de generar un comportamiento debido a sus limitaciones de sus acciones, Q-Learning no llegará a tener ese comportamiento complejo, sin embargo si esas acciones pueden ser comportamientos más complejos que una acción, entonces si podrá generar dichos comportamientos esperados, como en el caso de BADRL en el uso de otros comportamientos como en la generación de la triangulación a disparo a portería.

VI. CONCLUSIONES FINALES Y TRABAJO FUTURO DE LA TESIS.

Como se ha señalado anteriormente, el trabajo fundamental de esta tesis es realizar comportamientos por medio del aprendizaje por refuerzo. Para ello se generó una nueva herramienta conocida como BADRL (Behavior Agent Definition by Reinforcement Learning) que tiene la capacidad de generar dichos comportamientos por medio de tres archivos de configuración.

En general las ventajas de utilizar BADRL como lenguaje para generar comportamientos por medio de aprendizaje por refuerzo son las siguientes:

1. Capacidad de manejarse en cualquier tipo de plataforma que pueda conectarse con Java5.
2. Capacidad de manejar un número ilimitado de acciones.
3. Capacidad de manejar un número ilimitado de reglas de evaluación.
4. Capacidad de utilizarse en cualquier tipo de ambiente: interno o externo.
5. Capacidad de cambiar parámetros de configuración del algoritmo Q-Learning de manera simple y sencilla.
6. Generación sencilla de reportes por cada iteración del algoritmo.

1. CAPACIDAD DE MANEJARSE EN CUALQUIER TIPO DE PLATAFORMA QUE PUEDA CONECTARSE CON JAVA5.

Dentro de este panorama, la herramienta BADRL fue diseñada y creada con la intención de utilizarse en diferentes plataformas de programación. A pesar de que BADRL ha sido utilizado por vez primera dentro del campo de la simulación, la arquitectura computacional permite que se utilice en otras plataformas de la robótica como son los Sony AIBO™. Como se muestra en la figura 63, BADRL es un sistema basado en Java5 el cual debe tener la conexión con las librerías del sistema operativo del robot al cual se quiere analizar.

Un punto importante aquí radica principalmente en el uso de Java5, pues la mayoría de los robots no tienen la capacidad de tener las librerías de este lenguaje de programación, sin embargo la ventaja de esta herramienta también radica que se puede utilizar sobre un simulador de este robot. Actualmente en el mercado se cuenta con un software especializado de Microsoft conocido como Microsoft Robotics Studio [1]. Este programa tiene la capacidad de generar simuladores de diferentes robots físicos.

2. CAPACIDAD DE MANEJAR UN NÚMERO ILIMITADO DE ACCIONES.

Debido a que cada comportamiento necesita de tener un número de acciones que pueden variar, BADRL fue creado con el propósito de tener un archivo de configuración en el cual podrá leer cualquier cantidad de acciones posibles que el agente pueda realizar. En otras palabras, el número

de acciones posibles por utilizar en BADRL es el mismo que pueda utilizar el robot físico o simulado.

Un aspecto fundamental a señalar en este momento es que mientras más acciones utilice BADRL más tardará en llegar al comportamiento por aprender. Esto se debe a la misma esencia del algoritmo Q-Learning. Pues como se mencionó anteriormente, Q-Learning es un algoritmo que es capaz de explorar y explotar acciones de acuerdo a un parámetro de temperatura como lo es alfa. Dentro de este contexto si alfa es mayor explorará mucho más de lo explotado y si es este caso, y el número de acciones es grande, tardará en encontrar una relación en cada una de estas acciones para llegar al comportamiento.

3. CAPACIDAD DE MANEJAR UN NÚMERO ILIMITADO DE REGLAS DE EVALUACIÓN.

Generalmente, los ambientes de pruebas para los comportamientos para jugar fútbol de los agentes son dinámicos, esto es, una misma acción en un mismo punto dentro del campo de juego puede tener diferentes consecuencias, pues existen varios factores externos que modifican el resultado de esta acción.

Debido a lo anterior, BADRL fue diseñado para generar cualquier cantidad de reglas para evaluar estos ambientes dinámicos. El punto importante aquí es que esta capacidad depende de la manera en que el programador requiera evaluar de manera global estas respuestas, pues queda claro que el programador puede o no conocer todas las posibles respuestas del ambiente para una acción. Si el programador las conoce, BADRL tiene la capacidad de aceptar todas ellas y si no, el programador debe tener el esquema al menos global de lo que realmente quiere que haga el comportamiento y evaluarlo. En el peor de los casos, el programador puede configurar BADRL para indicar si el agente no llego a lo que se requiere y castigarlo solamente.

4. CAPACIDAD DE UTILIZARSE EN CUALQUIER TIPO DE AMBIENTE: INTERNO O EXTERNO.

Como se ha mencionado anteriormente, el ambiente de pruebas puede ser muy dinámico y BADRL tiene la capacidad para evaluar este ambiente. Esto es fundamental, pues BADRL es un sistema que depende de la evaluación del ambiente y la cantidad de acciones posibles y no del ambiente. Con esto se puede tomar como conclusión que no BADRL no toma en consideración el lugar del ambiente de pruebas. Generalmente las pruebas para jugar fútbol se pueden realizar en ambientes internos o externos.

Si se toma como consideración un ambiente de pruebas externo, esto es, con luz variable como jugar directamente con la luz solar, BADRL se maneja de la misma manera que si fuera dentro de ambientes de pruebas internos. Esto se toma como ventaja, pues la mayoría de los programas en el campo de la IA en robótica son totalmente en simulación o dependientes de una plataforma y si es así, dependen del hardware y del ambiente de pruebas.

5. CAPACIDAD DE CAMBIAR PARÁMETROS DE CONFIGURACIÓN DEL ALGORTIMO Q-LEARNING DE MANERA SIMPLE Y SENCILLA.

Q-Learning es un algoritmo que utiliza variables para poder llegar a un objetivo específico. De manera matemática, se utilizan dos variables para lograr que este algoritmo funcione. Como se puede ver en la fórmula siguiente, estas variables son: ALFA y GAMMA. Sin embargo, hay varias maneras que se puede realizar el uso de los valores Q. Para ello BADRL utiliza redes neuronales y el algoritmo Boltzmann. Las cuales tiene otras dos variables para lograr su objetivo, éstas son λ y τ [Figura 61].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a [Q(s_{t+1}, a_t)] - Q(s_t, a_t)\right]$$

Figura 59. Definición matemática del algoritmo Q-Learning.

Debido a lo anterior, BADRL fue diseñado para cambiar de manera simple, estos parámetros. De hecho, es vital para el funcionamiento esperado del sistema, que el programador cambie el archivo de configuración *conf.xml* que contiene la información de estos parámetros. En la figura se muestra este archivo con la información de los parámetros. [Figura 62]

```
<!--  qLearning parameters  -->
      <parameters>
            <alfa-param value="0.5"/>
            <gamma-param value="0.8"/>
            <lambda-param value="1"/>
            <boltzmann-param value="true"/>
            <temperature-param value="0.9"/>
      </parameters>
```

Figura 60. Uso del archivo *conf.xml* para modificar los variables utilizadas por Q-Learning.

6. TRABAJO FUTURO DE LA INVESTIGACIÓN.

Ahora bien, ya que se tiene definida las ventajas, falta analizar el trabajo futuro de esta tesis. Para entender el futuro sobre esta investigación es importante definir el campo en donde ésta actualmente se utiliza y los posibles trabajos futuros que encierran este trabajo.

En primera instancia, este trabajo de investigación fue implementado directamente sobre el simulador principal que utiliza el equipo TecRams como proyecto primordial del área de investigación en robótica en el Instituto Tecnológico y de Estudios Superiores de Monterrey en el Campus Estado de México. TecRams utiliza este simulador como herramienta para generar comportamientos básicos y complejos para agentes que juegan fútbol. Este simulador es utilizado con el fin de obtener estos comportamientos y no dañar los componentes físicos de los robots además de esta manera se puede anticipar el código implementado en el simulador antes de pasar al agente físico.

Con lo anterior, resulta claro que es fundamental el uso del simulador dentro del equipo. De hecho, cabe señalar que este simulador fue parte de la tesis de maestría de uno de los integrantes de éste. Como trabajo futuro en esta tesis se refirió la generación de comportamientos por medio de aprendizaje por refuerzo, de ahí salió en primera instancia la idea de la generación de esta tesis.

Ahora bien, ya que se tiene entendido la base del surgimiento de esta tesis, ahora toca el turno de explicar cómo se utiliza el simulador y el sistema de aprendizaje por refuerzo (BADRL). De manera general, el equipo utiliza BADRL para generar comportamientos básicos. Estos comportamientos son utilizados principalmente para los retos (challenges) de la liga de Robocup. Además genera varias maneras diferentes de comportamiento para el cumplimiento de un solo objetivo, en otras palabras, puede generar varios comportamientos para realizar una misma tarea, cosa que es utilizada y evaluada por el equipo de TecRams para meterlo como código principal en el comportamiento para jugar fútbol en los agentes físicos (Sony Aibo).

Ya que se tiene el panorama general del uso del simulador y BADRL. Ahora los requerimientos actuales del equipo TecRams se pueden clasificar de tres maneras distintas:

1. Generar comportamientos en donde varios jugadores aprendan un comportamiento colaborativo al mismo tiempo.
2. Generar comportamientos por aprendizaje por refuerzo utilizando otro algoritmo como Sarsa y encontrar patrones para mejorar BADRL.
3. Realizar pruebas de BADRL con otras plataformas de robots físicos como por ejemplo: Pionner 3atx, 3adx y los amigobots. [2]

A. GENERAR COMPORTAMIENTOS EN DONDE VARIOS JUGADORES APRENDAN UN COMPORTAMIENTO COLABORATIVO AL MISMO TIEMPO.

Este tema tiene como base el caso que se tuvo con la triangulación a disparo a portería, pues como se mencionó, el problema radica fundamentalmente en la generación de un comportamiento colaborativo para dos agentes, sin embargo esta tesis trató de resolverlo por medio del uso de *BADRL* primero para la generación del comportamiento del jugador J2 *(Ver Capítulo 6.6)*. Este jugador recibe y envía a un punto dado el esférico sin moverse de su posición. Por otro lado el jugador J1 aprendió por medio de BADRL a enviar el esférico, caminar y/o correr y recibir el esférico del jugador J2 para después disparar a portería.

En el caso anterior, fue claro que se utilizó BADRL como una herramienta para la creación de comportamientos pero de manera singular, esto es, a pesar de que en el medio se utilizaban dos agentes, BADRL fue utilizado dos veces, uno por cada agente, lo que provoca que el comportamiento sea colaborativo más no en coordinación con otros agentes. Este tema precisamente es un trabajo futuro de esta investigación pues al parecer existen nuevas actualizaciones del algoritmo Q-Learning en donde involucran más agentes para aprender algo en común. Por lo que si es necesario que se construya ese algoritmo, se puede modificar el sistema BADRL para recibir estos cambios de fondo y de esta manera generar comportamientos con varios agentes que aprendan al mismo tiempo en un modelo del medio determinado como es el caso del juego de fútbol.

B. GENERAR COMPORTAMIENTOS POR APRENDIZAJE POR REFUERZO UTILIZANDO OTRO ALGORITMO COMO SARSA Y ENCONTRAR PATRONES PARA MEJORAR BADRL.

La meta principal de esta investigación fue la generación de comportamientos para agentes que juegan fútbol por medio de Q-Learning. Esta aproximación se dio en primera instancia por la

necesidad del equipo TecRams de generar nuevas herramientas para generar comportamientos y segundo el desenvolvimiento que trae métodos de aprendizaje por refuerzo para generar dichos comportamientos.

Dentro del contexto anterior, se determinó sólo utilizar el algoritmo de Q-Learning, sin embargo como trabajo futuro queda definitivamente el uso de otros métodos de diferencia temporal como Sarsa pues de acuerdo a las investigaciones hechas por Peter Stone y Manuela Veloso, Sarsa tiene un comportamiento muy distinto al largo plazo, lo cual podría indudablemente repercutir en el uso de la herramienta BADRL. De tal suerte que es necesario determinar y utilizar este algoritmo para revisar, comparar y analizar si existen mejoras sobre este trabajo realizado.

C. REALIZAR PRUEBAS DE BADRL CON OTRAS PLATAFORMAS DE ROBOTS FÍSICOS COMO POR EJEMPLO: PIONNER 3ATX, 3ADX Y AMIGOBOTS.

De acuerdo a la investigación de esta tesis, el uso de aprendizaje por refuerzo (Q-Learning) para la generación de comportamientos para agentes que juegan fútbol, se desarrolló el sistema BADRL que puede ser utilizado y modificado para agentes físicos, sin embargo para las pruebas realizadas no se implementó directamente sobre estos agentes, sino que se utilizó sobre un simulador *XML Behaviour Control* para ello [Figura 63].

Por lo anterior, como trabajo futuro, es necesario que se desarrolle e implemente esta herramienta con robots físicos como por ejemplo el Pionner 3ATX, 3ADX, Amigobots o incluso los mismos robots Sony AIBO. Este tipo de robots son investigados generalmente en universidades que se enfocan al estudio primordial de la robótica móvil, incluso actualmente en el Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Estado de México se cuentan con estos robots por lo que puede llegar a ser bastante útil el uso de la herramienta BADRL para estos agentes dinámicos.

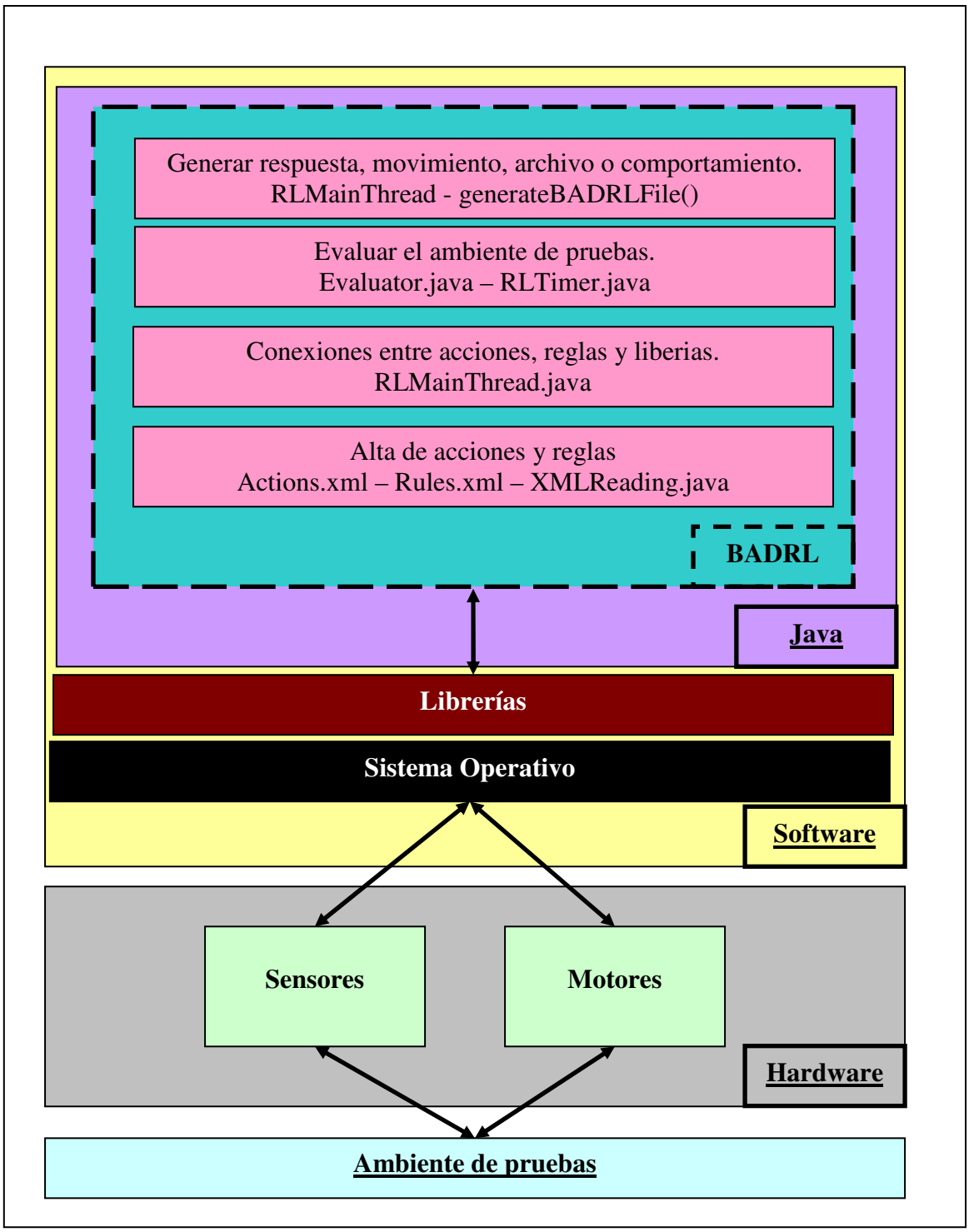


Figura 61. Partes y módulos principales del desarrollo interno de la herramienta *BADRL*.

VII. REFERENCIAS Y FUENTES CONSULTADAS.

[1] AIBO SDE official web site [en línea], Sony Corporation, 2006, (29 de enero de 2006). Disponible en Web: http://openr.aibo.com

[2] BALCH Tucker, "Integrating RL and Behavior-based Control for Soccer". Proceedings of the IJCAI Workshop on RoboCup, Nagoya, Japón. 1997.

[3] COPPIN B. *Artificial Intelligence Illuminated*. Primera edición, Estados Unidos de América: Jones and Bartlett Publishers. 2004.

[4] Free Connectionist Q-Learning Java Framework [en línea]. (26 de Julio de 1997). Disponible en Web: http://elsy.gdan.pl/.

[5] GermanTeam : Home [en línea]. GermanTeam Developer Team, 2002, (7 de noviembre de 2006). Disponible en Web: http://www.germanteam.org.

[6] Información sobre robots móviles. Disponible en: http://www.mobilerobots.com Fecha de consulta: 12 de Noviembre de 2007.

[7] KIM John-Hwan, Prahlad, V.; "Multi-Agent Systems: A Survey from the Robot-soccer Perspective". Int. J. Intelligent Automation and Soft Computing. 2000, pp. 3-17.

[8] MATSUBARA H., Noda I. Hiraki K. "Learning of Cooperative actions in multi-agent systems: a case study of pass play in Soccer". The 9th International Conference on Intelligent Autonomous Systems (IAS-9). 2006, p. 3.

[9] Microsoft Robotics Studio. Información de Microsoft Robotics Studio. Disponible en: http://www.microsoft.com/spanish/msdn/articulos/archivo/050207/voices/learn_default.mspx. Fecha de consulta: 21 de Octubre de 2007.

[10] OGINO M., Katoh Y,, Aono M., Asada M., Hosoda K., "Reinforcement Learning of Humanoid Rhythmic Walking Parameters based on visual Information". The 9th International Conference on Intelligent Autonomous Systems (IAS-9). 2006, p. 85.

[11] Proyecto de los Robots Cuadrúpedos [en línea]. Instituto Tecnológico y de Estudios Superiores de Monterrey. Campus Estado de México, 2002, (octubre 2006). Disponible en Web: http://www.cem.itesm.mx/robocup

[12] Robocup Official Site [en línea], The Robocup Federation. 2006, (22 de octubre de 2006). Disponible en Web: http://www.robocup.org/

[13] RoboCup-Rescue Official Web Page [en línea], The Robocup Federation, 2001, (4 de junio de 2006). Disponible en Web: http://www.rescuesystem.org/robocuprescue/

[14] RUSSEL, S. and NORVING, P., *Artificial Intelligence, A Modern Approach*, Segunda Edición, Prentice Hall, 2003. 938 p.

[15] SALUSTOWICZ Rafal, P., Wiering, M., Schmidhuber, J; "Learning Team Strategies: Soccer Case Studies". Machine Learning 33. Kluwer Academic Publishers. 1998, pp. 263-282.

[16] Sony Global- AIBO Global Link [en línea]. Sony Corporation, 2006, (2 de abril de 2006). Disponible en Web: http://www.sony.net/Products/aibo/

[17] STONE P., Sutton S. R.,, KuhlmannG. "Reinforcement Learning for *Robocup* Soccer Keepaway". The 9th International Conference on Intelligent Autonomous Systems (IAS-9). 2006, p. 85.

[18] STONE P., Veloso M., "Team-Partitioned, Opaque-Transition Reinforcement Learning". The 9th International Conference on Intelligent Autonomous Systems (IAS-9). 2006, p. 85.

[19] SUTTON R, Barto A. G. *Reinforcement Learning. An Introduction*. Primera edición. Estados Unidos de América: The MIT Press. 1998. 322 p.

[20] Técnicas de fútbol [en línea]. Tácticas de Futbol. 2006, (30 de junio de 2007). Disponible en Web: http://www.solofutbolnet.com/06_tecnicas.htm

[21] TAPIA J.J., Reyes-Rico C., Ramírez J. "Automatic behavior generation in a multi agent system through genetic programming". 3rd LatinAmerican Robotics Symposium, Santiago, Chile, 2006.

[22] Tekkotsu/aperios [en línea]. Carnegie Mellon University. 2002 (noviembre de 2006). Disponible en Web: http://cvs.tekkotsu.org/cgi/viewcvs.cgi/Tekkotsu/aperios/

[23] VEGA J.L., Ramírez, J., Junco, A; "Major behavior definition of football agents through XML". The 9th International Conference on Intelligent Autonomous Systems (IAS-9). 2006, p. 85.

[24] WebHome – Website – Four-Legged League [en línea]. Robocup Four-Legged League, 2002 (10 de noviembre de 2006). Disponible en Web: http://www.tzi.de/4legged/bin/view/Website/WebHome .

[25] XABSL. The Extensible Agent Behavior Specification Language [en línea]. XABSL Developer Team, 2002, (29 de septiembre de 2006). Disponible en Web: http://www2.informatik.hu-berlin.de/ki/XABSL

VIII. ANEXOS.

1. EJEMPLO DEL LENGUAJE QUE UTILIZA *XABSL*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<option xmlns="http://www.ki.informatik.hu-berlin.de/XABSL2.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ki.informatik.hu-berlin.de/XABSL2.2
../../../../Tools/Xabsl2/xabsl-2.2/xabsl-2.2.option.xsd" name="aproachToBall"
initial-state="adelante">
       &ball-symbols;
       &kick-selection-symbols;
       &math-functions;
       &motion-request-symbols;
       &angle-symbols;
       &strategy-symbols;
       &common-basic-behaviors;
       &simple-basic-behaviors;
       &options;

<state name="adelante">
      <subsequent-basic-behavior ref="walk" />
      <set-output-symbol ref="head-control-mode"
            value="head-control-mode.search-for-ball"/>
      <decision-tree>
      <if> <condition description="TecRams parsed condition">
            <and>
                  <greater-than>
                  <decimal-input-symbol-ref ref="ball.seen.angle"/>
                  <decimal-value value="30"/>
                  </greater-than>
                  <and>
                        <greater-than>
                              <decimal-input-symbol-ref
                        ref="ball.seen.angle"/>
                        <decimal-value value="180"/>
                        </greater-than>
                        <less-than>
                              <decimal-input-symbol-ref
                        ref="ball.seen.angle"/>
                        <decimal-value value="360"/>
                        </less-than>
                  </and>
            </and>
            </condition>
                  <transition-to-state ref="turnRight"/>
      </if>
      <else>
            <transition-to-state ref="adelanteRight"/>
      </else>
      </decision-tree>
</state>
</option>
```

2. EJEMPLO DEL LENGUAJE QUE UTILIZA *XML BEHAVIOUR CONTROL*.

```
<?xml version="1.0"?>
<!DOCTYPE behaviorSet SYSTEM "file:behavior.dtd">

<behaviorSet>
<behavior name="searchApproachKickBall" isInitialBehavior="true">

      <state name="adelante" iterate="true">
            <action id="MOTION-goForward"/>
            <action id="COMPLEX-headToBall"/>

            <transition state="acomodate">
                  <event name="VISION-shootDistance"/>
            </transition>

            <transition state="moveBodyRight">
                  <analogEvent value="30" name="angleToBall"/>
                  <event name="VISION-ballAtRight"/>
            </transition>

            <transition state="moveBodyLeft">
                  <analogEvent value="270" name="angleToBall"
                  operator="less"/>
                  <event name="VISION-ballAtLeft"/>
            </transition>

            <transition state="searchLeft">
                  <event name="VISION-ballAtLeft"/>
                  <event name="VISION-ballFound" operator="not"/>
            </transition>

            <transition state="searchRight">
                  <event name="VISION-ballAtRight"/>
                  <event name="VISION-ballFound" operator="not"/>
            </transition>

            <transition state="searchRight">
                  <event name="VISION-ballFound" operator="not"/>
            </transition>
      </state>

</behavior>
</behaviorSet>
```

3. EJEMPLO DE CONFIGURACIÓN DEL ARCHIVO *conf.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<qlearning-app name="XML Behavior Control with BADRL" version="1.0"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="configurationSchema.xsd">

      <!--  qLearning parameters  -->
      <parameters>
            <alfa-param value="0.5"/>
            <gamma-param value="0.8"/>
            <lambda-param value="1"/>
            <boltzmann-param value="true"/>
            <temperature-param value="0.9"/>
      </parameters>

      <!--  qLearning rewards -->
      <rewards>
            <movingWell-reward value="-0.2" />
            <collides-reward value="-0.2"/>
            <default-reward value="-0.5" />
      </rewards>
</qlearning-app>
```

4. EJEMPLO DE CONFIGURACIÓN DEL ARCHIVO *actions.xml*.

```
<qlearning-actionSet name="Acciones para JLV" version="1.0"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="actionSchema.xsd">
      <ql-preconditions language="xml" action="action"
                        transition="transition" />

      <ql-action name="camina_lento" isText="true" iterate="false"
                 action="id='MOTION-goForward' param1='2'">
            <ql-transition name="event">
                        name="VISION-ballVeryNear",
                        name="VISION-shootDistance" operator="not"
            </ql-transition>
      </ql-action>

      <ql-action name="camina_rapido" isText="true" iterate="true"
              time="1000" action="id='MOTION-goForward' param1='8'">
            <ql-transition name="event">
                        name="VISION-ballNear",
                        name="VISION-shootDistance" operator="not"
            </ql-transition>
      </ql-action>

      <ql-action name="tira_derecha" isText="true" iterate="false"
            action="id='MOTION-shootFront' param1='400' param2='-30'">
            <ql-transition name="event">
                        name="VISION-shootDistance",
                        name="VISION-ballAtLeft"
            </ql-transition>
      </ql-action>

      <ql-action name="tira_izquierda" isText="true" iterate="false"
            action="id='MOTION-shootFront' param1='400' param2='30'">
            <ql-transition name="event">
                        name="VISION-shootDistance",
                        name="VISION-ballAtRight"
            </ql-transition>
      </ql-action>

      <ql-action name="tira_derecha_leve" isText="true"
            iterate="false" action="id='MOTION-shootFront'
                  param1='100' param2='-19'">
            <ql-transition name="event">
                        name="VISION-shootDistance",
                        name="VISION-ballAtLeft"
            </ql-transition>
      </ql-action>

      <ql-action name="tira_izquierda_leve" isText="true"
            iterate="false" action="id='MOTION-shootFront'
                  param1='100' param2='10'">
            <ql-transition name="event">
                        name="VISION-shootDistance",
                        name="VISION-ballAtRight"
            </ql-transition>
      </ql-action>
</qlearning-actionSet>
```

5. GENERACIÓN DE NUEVOS COMPORTAMIENTOS PARA EL PROYECTO *robocup4* EN EL SIMULADOR *XML BEHAVIOUR CONTROL*.

Actualmente BADRL se está desarrollando sobre el XML Behaviour Control. Éste se encuentra dentro de la ruta: `/robocup4/rl`.

La clase principal que se debe ejecutar para observar la generación de comportamientos por medio de aprendizaje por refuerzo es MainRLClass.java que se encuentra inmediatamente debajo de la ruta descrita anteriormente.

Los dos archivos de configuración se encuentran en la ruta `/robocup4/rl/conf`. Cada archivo tiene su respectivo *schema*. El archivo *actions.xml* se valida por medio del archivo *actionSchema.xsd* y el archivo *conf.xml* por medio del *configurationSchema.xsd*.

El archivo final que se obtiene con el comportamiento aprendido se encuentra en la ruta `robocup4/rl/finalBehavior`. Este archivo es el resultado final de las operaciones de BADRL y contiene el compartimiento aprendido por un agente.

Para generar nuevos comportamientos sólo es necesario cambiar los archivos de configuración. El principal es el *actions.xml* que sólo debe contener el arreglo de acciones que el programador necesite que use el agente para generar el comportamiento.

6. INSTALACIÓN DE LA APLICACIÓN.

Este anexo tiene como principal objetivo la instalación correcta del programa BADRL para el simulador XML Behaviour Control. Cabe señalar que durante la realización de esta tesis, se tuvo la oportunidad de subir a un espacio dentro del portal del I.T.E.S.M. CEM ambos programas. Los cuales durante la lectura de esta tesis, el lector tendrá la oportunidad de bajarlos.

Las partes principales de instalación son las siguientes:
A. Instalación del ambiente gráfico de programación, Eclipse.
B. Instalación del XML Behaviour Control y BADRL.
C. Generación de comportamientos y uso del simulador con BADRL.

A. INSTALACIÓN DEL AMBIENTE GRÁFICO DE PROGRAMACIÓN, ECLIPSE

1. Bajar el ambiente gráfico de la página principal de Eclipse.

El programa de XML Beheavior Control y BADRL fueron desarrolados en el ambiente de programación Eclipse. Este programa hoy día es el más utilizado para desarrollar programas en el lenguaje de programación Java 5. Durante el desarrollo de esta tesis, éste fue cambiando de versiones. Sin embargo la última versión que fue utilizada es la conocida como Eclipse Europa. Se puede descargar de la siguiente liga: http://www.eclipse.org/downloads/

En la figura 64 se puede visualizar el sitio en el cual se puede descargar el programa, la opción es Eclipse IDE for Java Devolopers, pues a pesar de que el tanto el simulador y BADRL usan componentes de EJB, es recomendable bajar sólo la version Java Devolopers.

Por otro lado, si el sitio fue cambiado por alguna situación es recomendable bajar este ambiente de esta versión en la siguiente liga:
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/20070927/eclipse-java-europa-fall-win32.zip

2. Instalación del ambiente en la computadora personal del programador.

Una vez bajado el programa Eclipse dentro del disco duro del ordenador del programador, será necesario ir a la carpeta donde se bajó el programa y descomprimir el archivo. Generalmente el archivo que se baja es en formato .ZIP por lo que será necesario descomprimir archivo con la herramienta de Windows dando clic derecho sobre el ícono del programa bajado y dar clic en extraer aquí como se puede ver en la figura 65. Si Windows no tiene la opción de descomprimir, por lo tanto se tendrá que bajar un programa de ZIP como por ejemplo WinZip o WinRar disponibles en línea en la página siguiente: www.downloads.com.

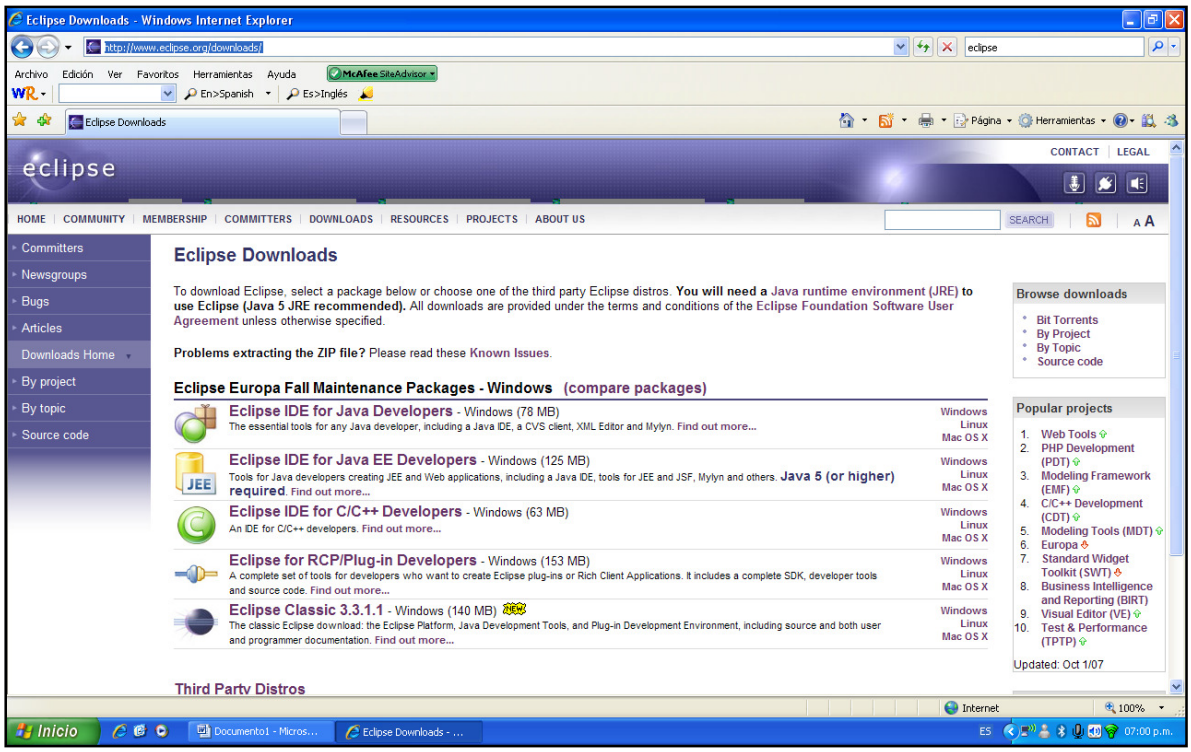


Figura 62. Sitio principal para bajar el programa de ambiente gráfico: Eclipse.

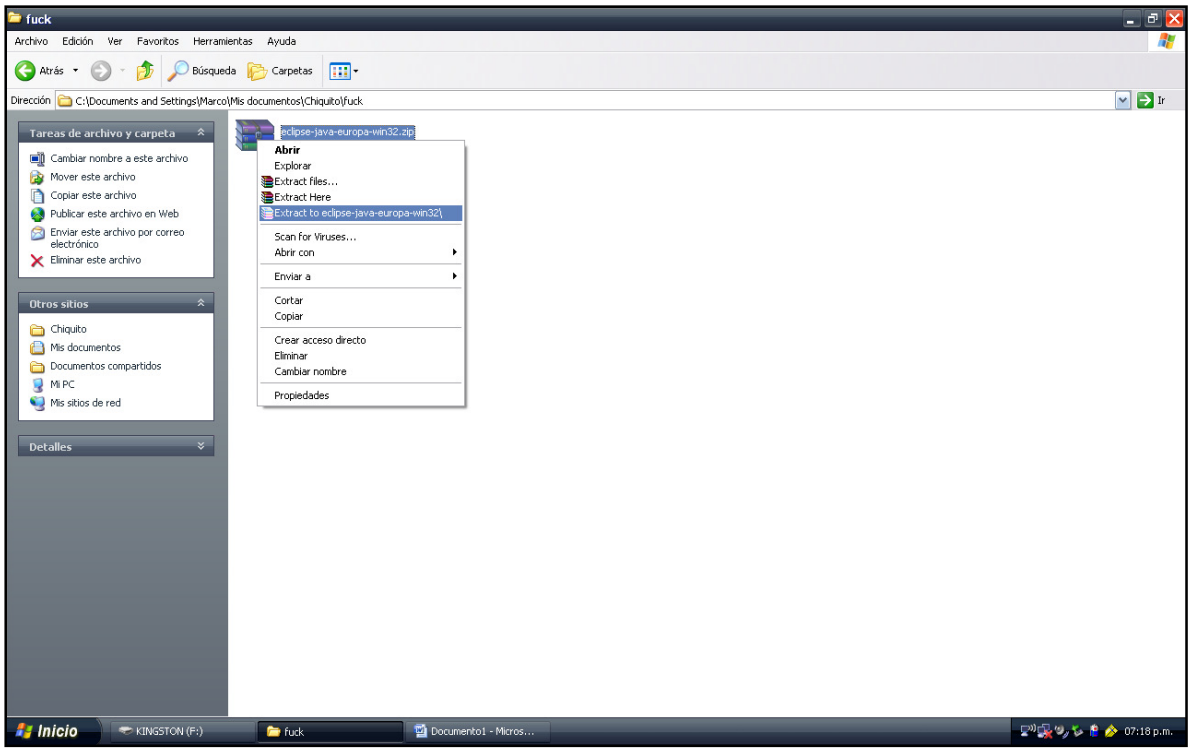


Figura 63. Ejemplo para descomprimir Eclipse en el ordenador.

3. Abrir y ejecutar el programa Eclipse

Dado que Eclipse no contiene un programa ejecutable de intalación, el programador tendrá que abrir el programa directamente del fólder descomprimido anteriormente. Para ello tendrá que abrir la carpeta descomprimida y buscar el programa ejecutable conocido como Eclipse.exe. En la figura 66 se puede observar que se ha seleccionado el ícono eclipse.exe el cual tendrá que ser ejecutado dándole doble clic para iniciar el programa.

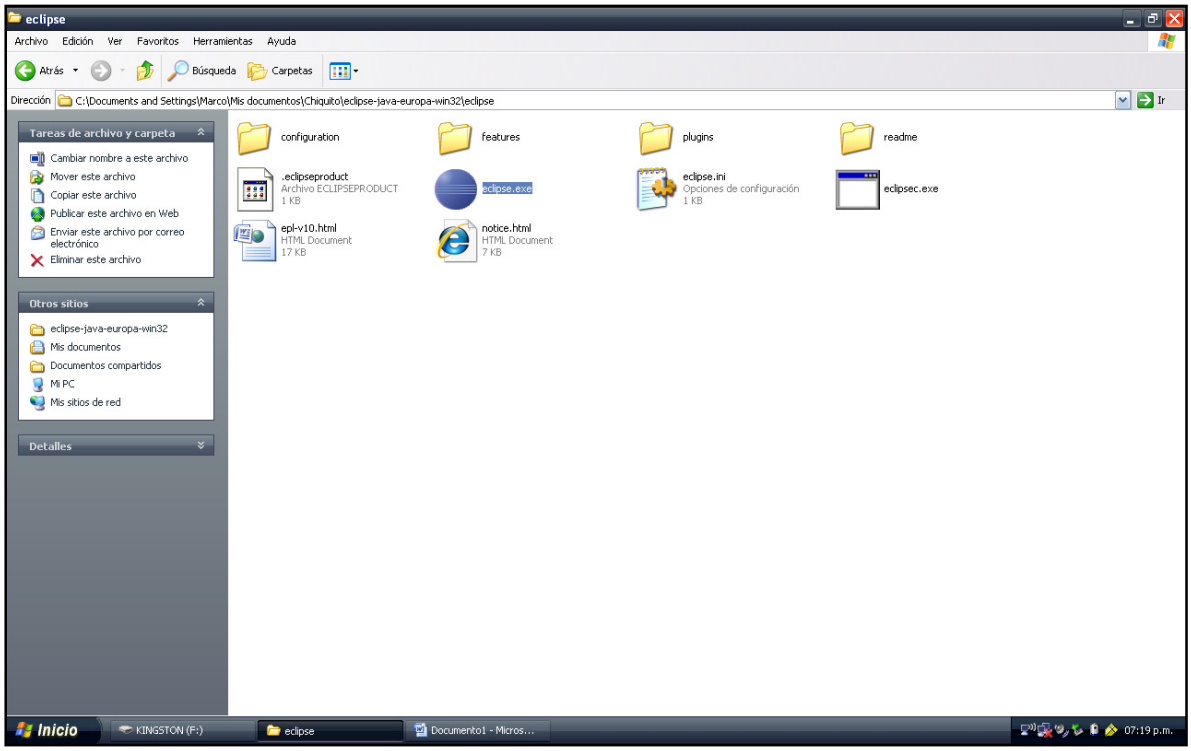


Figura 64. Selección del ícono eclipse.exe dentro de la carpeta de Eclipse.

4. Crear el espacio de trabajo (workspace) en Eclipse para instalar XML Behaviour Control y BADLR.

En el momento que el programador dá doble clic sobre el ícono de eclipse.exe, aparecerá en primera instancia la búsqueda del espacio de trabajo. Un espacio de trabajo es definido como el lugar físico dentro del disco duro del ordenador por default el cual tendrá los programas a ser usados dentro del ambiente. Por lo tanto es importante apuntar este espacio de trabajo en un lugar en el cual el programador tenga sus programas generados. En el caso particular de esta tesis, se apuntó en la carpeta C:\workspace como se puede ver en la figura 67.

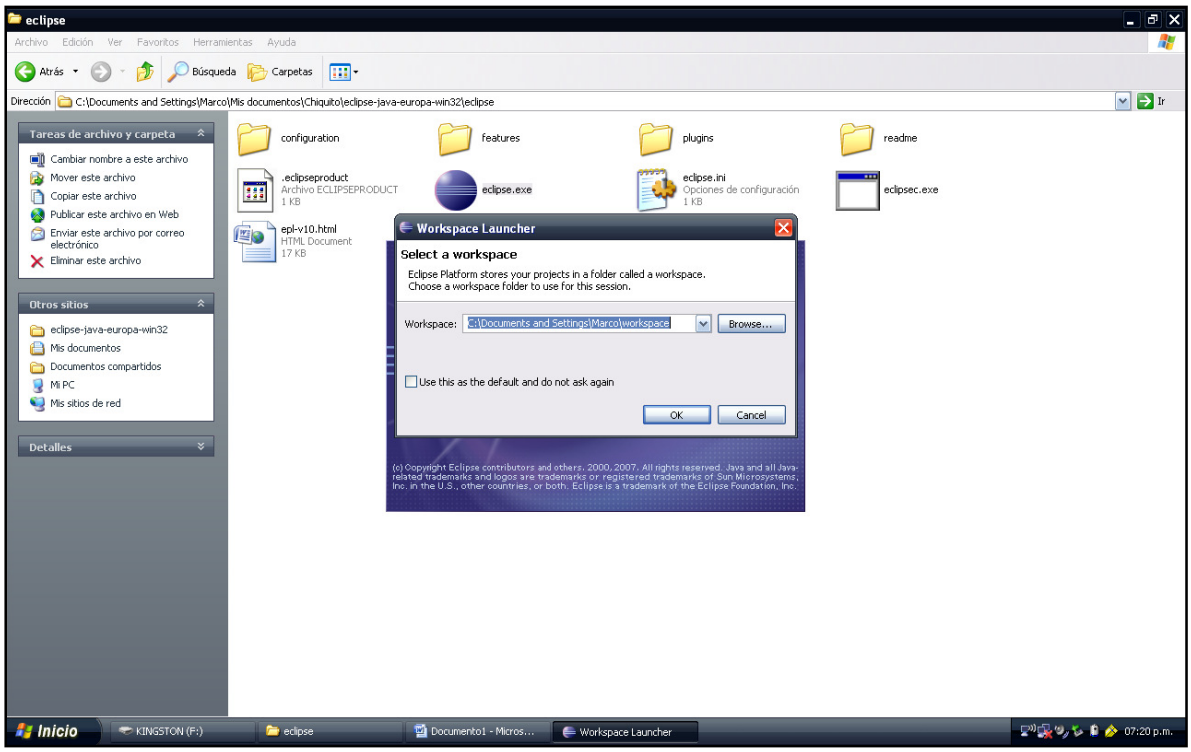


Figura 65. Ventana principal para definir el espacio de trabajo en Eclipse.

B. INSTALACIÓN DEL XML BEHAVIOUR CONTROL Y BADRL.

5. Descomprimir archivo robocup4.zip

Una vez instalado y generado el espacio de trabajo de Eclipse, el programador tendrá que bajar el simulador y el trabajo de desarrollo de esta tesis. Para ello, podrá bajar de la liga http://www.cem.itesm.mx/robocup/Equipo/carlos/robocup4.rar. Ahora bien, es importante señalar que lo que se acaba de bajar es la combinación de XML Behaviour Control (simulador) y BADRL (trabajo de esta tesis). Se tendrá que descomprimir el archivo robocup4.zip de igual manera que se descomprimió Eclipse en la sección anterior A2 como se muestra en la figura 68.

6. Copiar folder robocup4 al workspace creado

Una vez que se descomprime el simulador, se tendrá que copiar la carpeta completa al espacio de trabajo (workspace) creado en la sección A4. En el caso concreto de esta tesis, se copió esta carpeta en el directorio **C:\workspace**

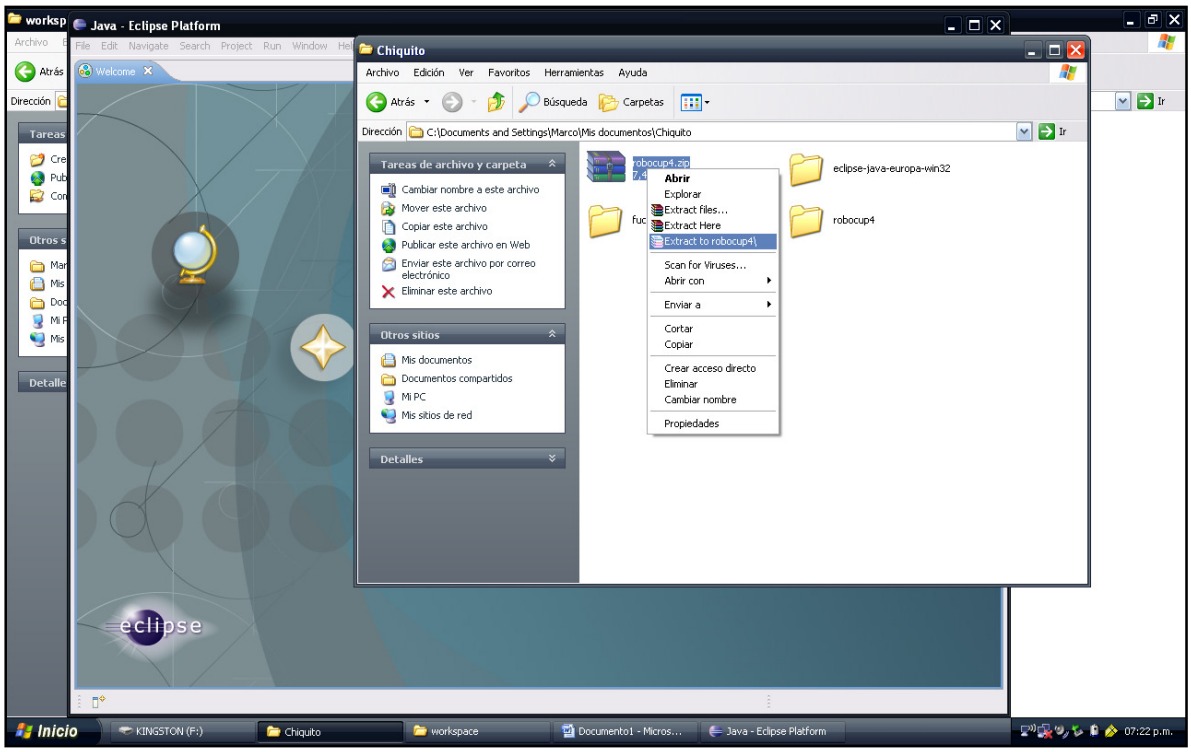


Figura 66. Descomprimir el archivo robocup4 con el simulador XML Behaviour Control.

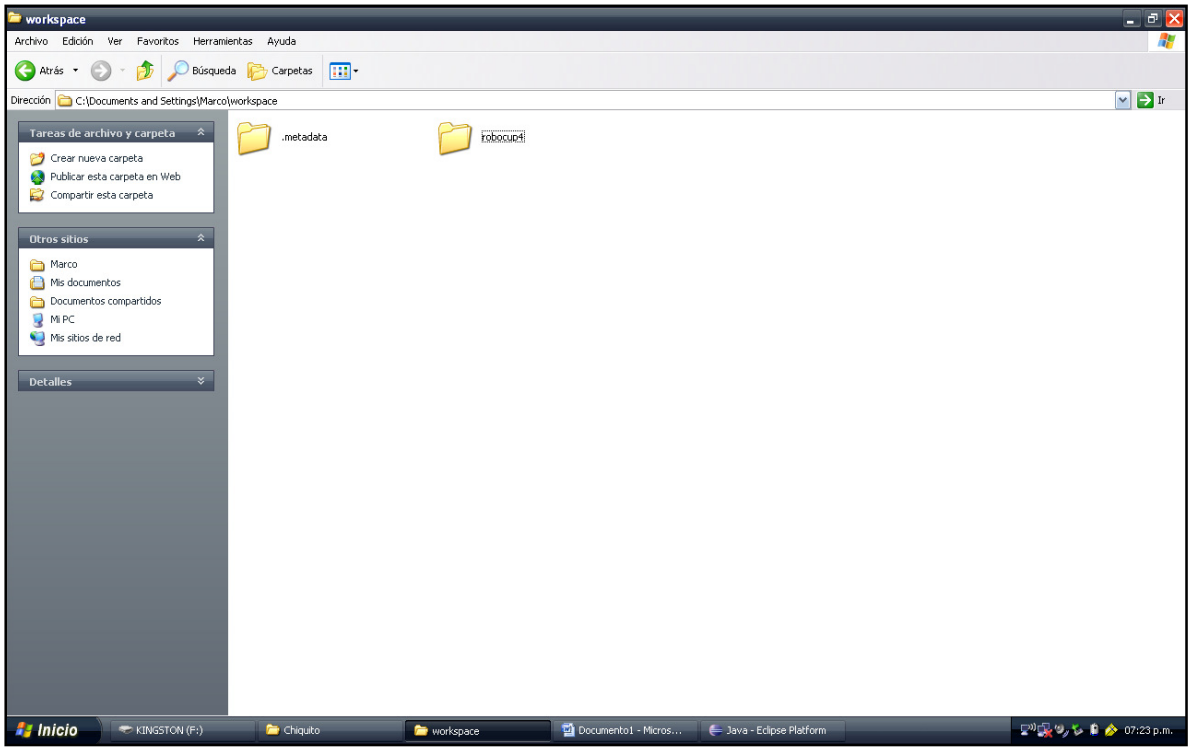


Figura 67. Ejemplo para copiar el folder que contiene el código del simulador y BADRL al espacio de trabajo por default creado.

7. Cargar robocup4 como workspace en Eclipse

Ya se copió el folder del simulador y BADRL al espacio de trabajo, será necesario cargar el código en el ambiente gráfico de programación Eclipse, para ello, el programador tendrá que dar clic en el menú principal en la sección File. De ahí saldrá un submenú y se tendrá que dar clic ahora en la palabra Import como se muestra en la figura 70.

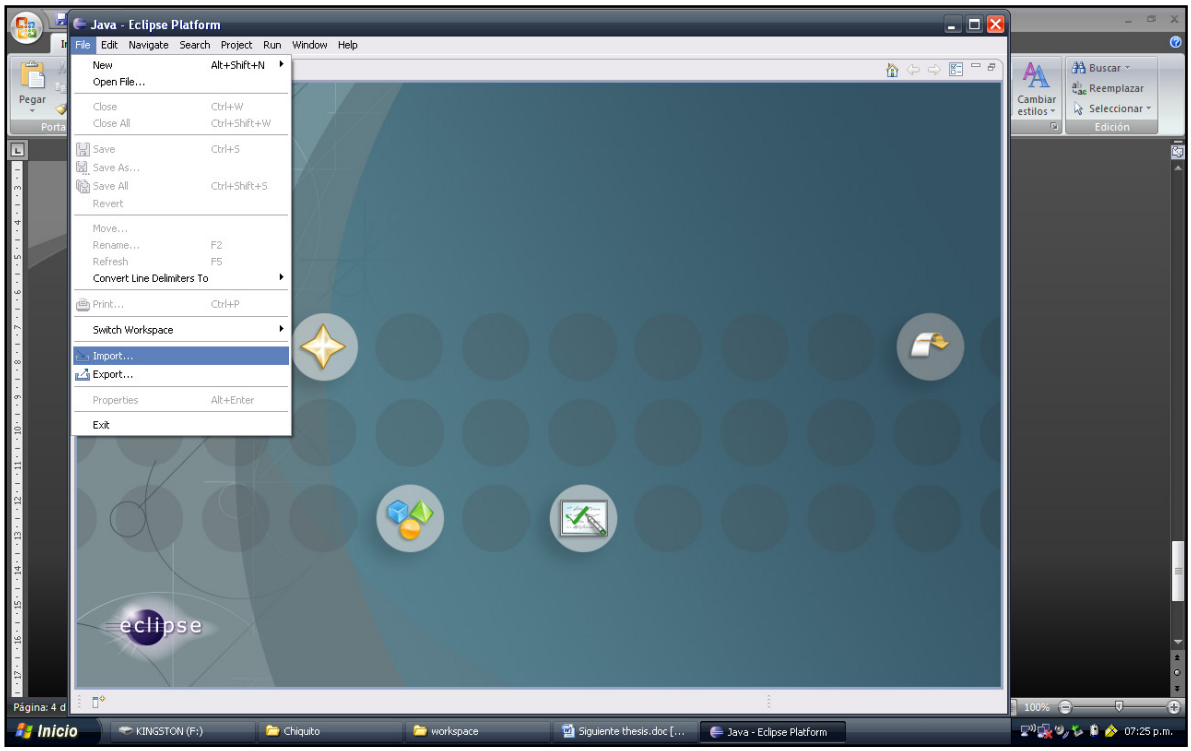


Figura 68. Menú del programa Eclipse para importar el código del espacio de trabajo.

8. Cargar robocup4 del workspace

En la siguiente ventana después de dar clic sobre File/Import del menú de Eclipse, el programador tendrá que seleccionar la opción conocida como *Existing Projects into Workspace* como se muestra en la figura 71. Esto servirá para indicarle a Eclipse dónde tendrá que cargar el código.

9. Buscar workspace en el directorio de Windows

Ya que el programador seleccionó la opción anterior, ahora tendrá que dar clic sobre la carpeta que contiene el código del simulador. En el caso de esta tesis está en C:\workspace y es el folder conocido como robocup4 como se muestra en la figura 72. Después de ello sólo será necesario dar Aceptar y Finalizar.

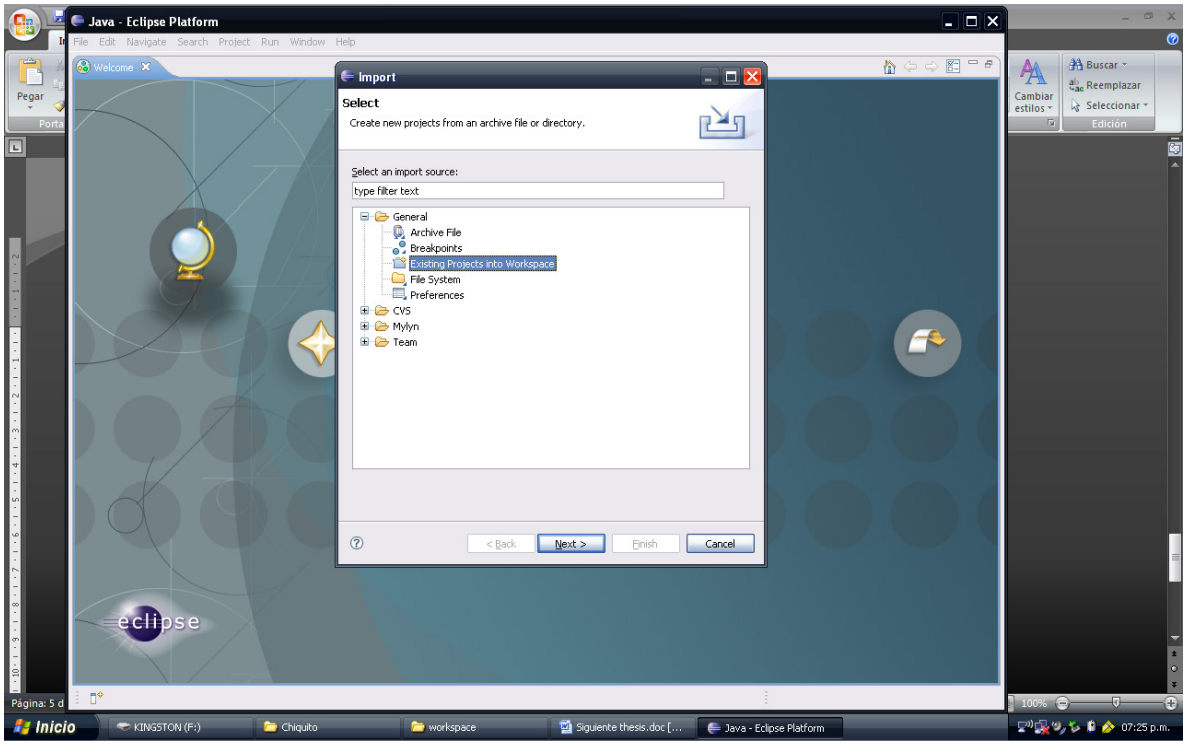


Figura 69. Ejemplo para importar el código del espacio de trabajo.

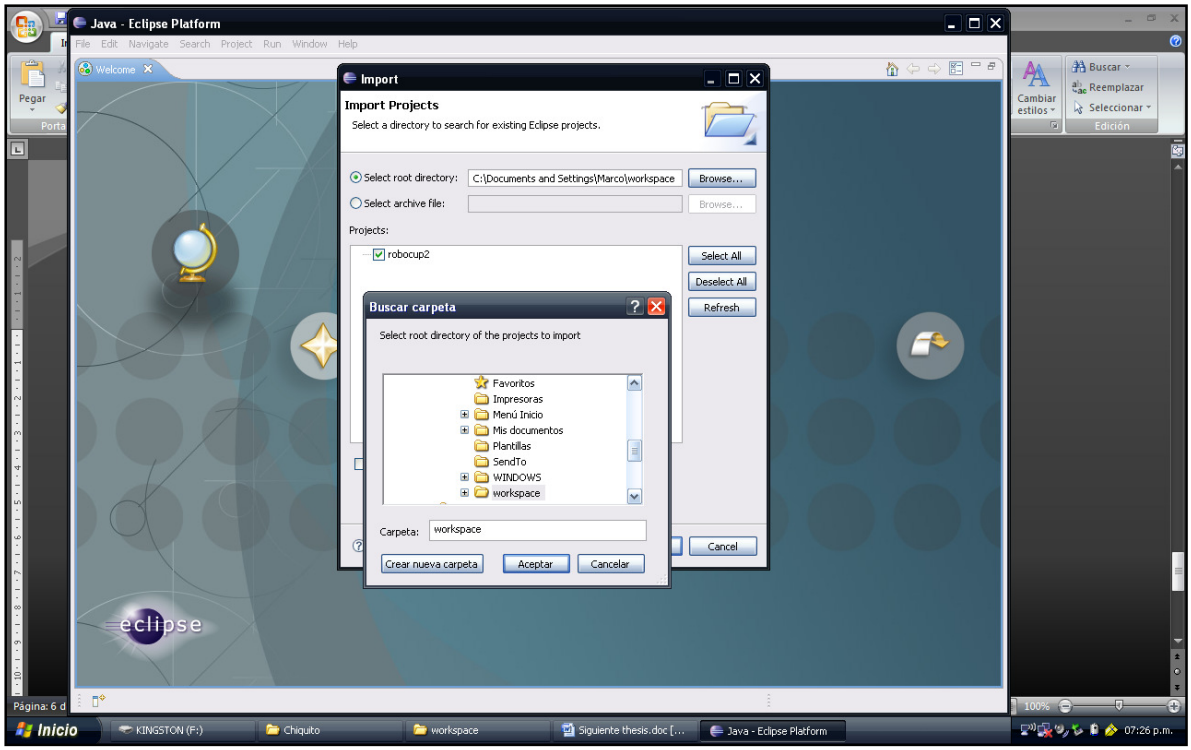


Figura 70. Ejemplo para buscar la carpeta robocup4 en el espacio de trabajo en Eclipse.

10. Cerrar pantalla Welcome

La primera vez que se instala Eclipse, aparecerá la ventana de Welcome, el cual tendrá que cerrarse dando clic sobre la cruz que se ubica al lado derecho de la palabra Welcome en una etiqueta por debajo del menú principal de Eclipse.

11. Abrir código robocup2

Ya que se ha cargado correctamente el código del simulador y BADLR y se cerró la pantalla de Welcome, se tendra que abrir el proyecto conocido como robocup2 en el panel izquierdo del ambiente de programación. Como se puede visualizar en la figura 73, existen varios proyectos y carpetas dentro de robocup2. En términos generales, existen 4 proyectos principales:

- **biology/genetic:** contiene el código principal de algoritmos genéticos sobre el simulador.
- **rl (reinforcement learning):** contiene el código principal de BADRL sobre el simulador.
- **parser:** contiene el código principal para convertir los códigos de los comportamientos generados del simulador al lenguaje XABSL.
- **xbdEngine y world:** contiene el código principal del simulador XML Behaviour Control.

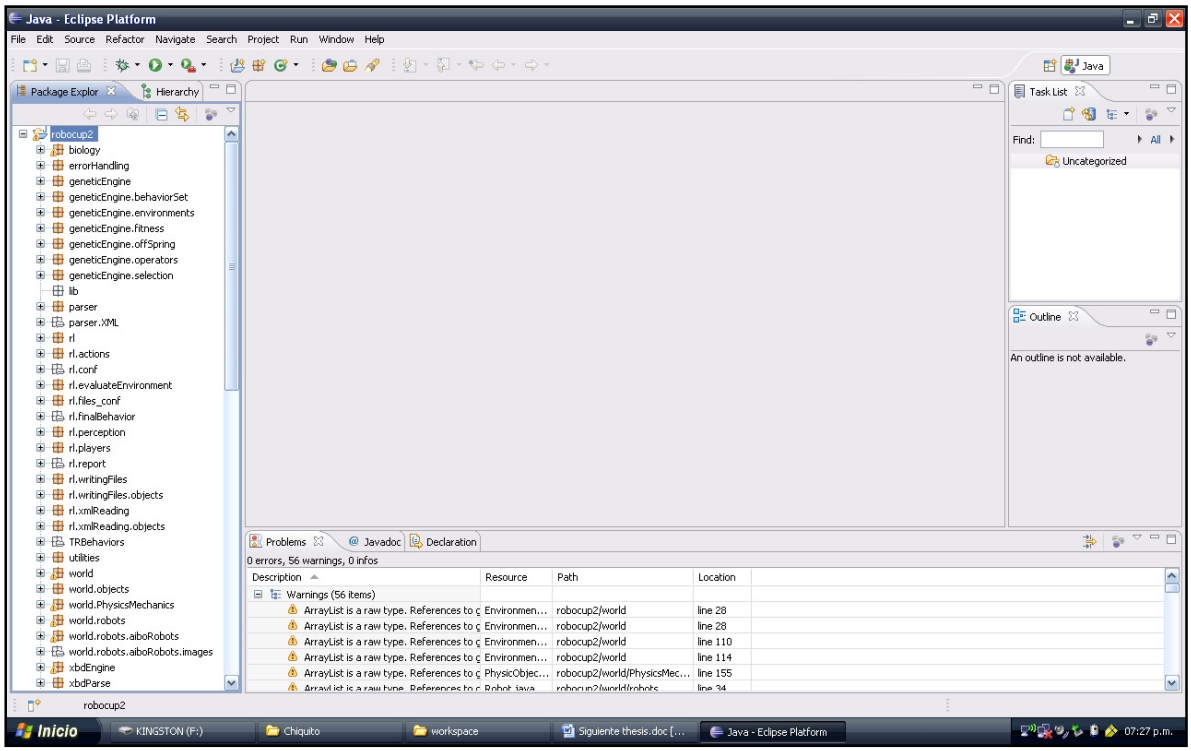


Figura 71. Visualización de los archivos y organización del código del XML Behaviour Control.

12. Abrir robocup2/rl.files_conf la clase FileConf.java; Modificar Files.CONF_FILE, Files.ACTIONS_FILE, Files.DTD_FILE y Files.RULES_FILE

Debido a que existen apuntadores y referencias a archivos de manera local, el programador tendrá que modificar la clase **FileConf.java** que se ubica en *robocup2/rl.files_conf*. Se tendrá que colocar de manera obligatoria la nueva referencia de los 4 archivos principalemente: conf.xml, actions.xml, behavior.dtd y rules.xml. Para ello tendrá que cambiar la enumeración principal y pondra la dirección física completa con diagonales invertidas.

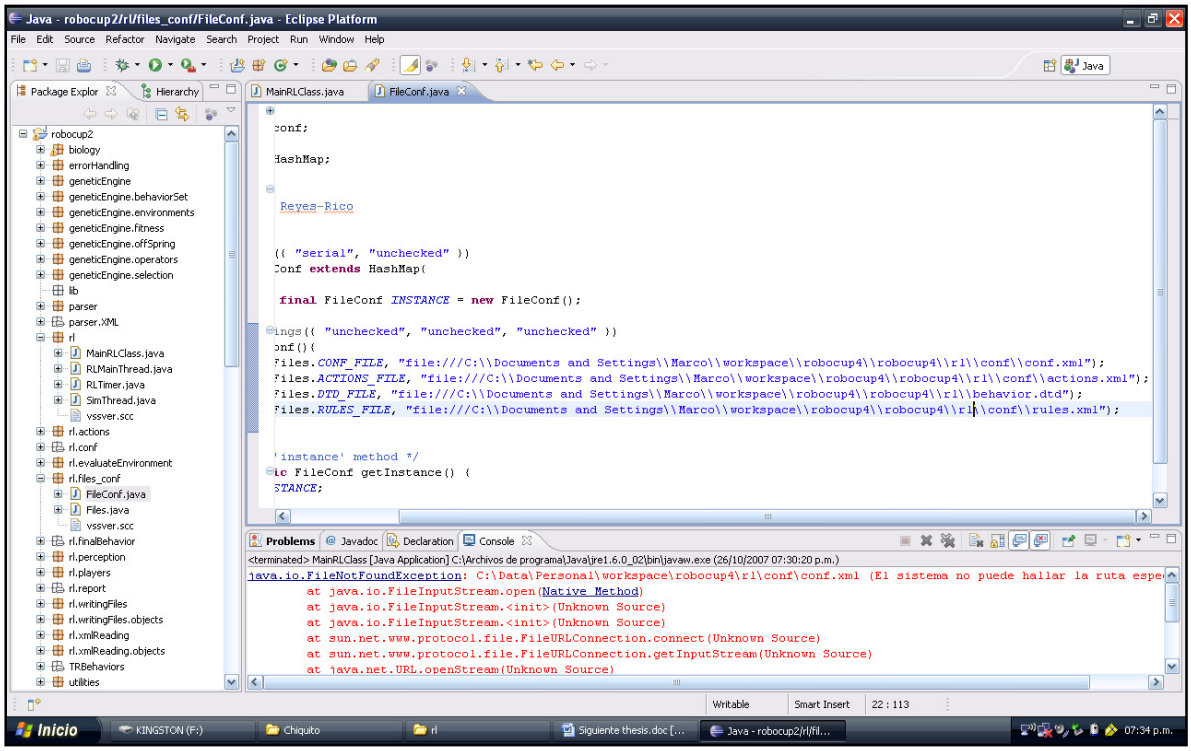


Figura 72. Ejemplo para cambiar la referencia de todos los archivos de BADRL.

13. Abrir robocup2/rl y dar clic botón derecho sobre MainRLClass.java

Ya que se configuró adecuadamente todo, ahora se dispondrá el programador a correr el programa completo, para ello tendrá que dar clic con botón derecho sobre el archivo MainRLClass.java. Dar clic sobre Run As e inmediatamente Run as Java Application.

14. Disfrutar el comportamiento y empezar la investigación

Ya que se configuró adecuadamente todo, ahora se dispondrá el programador a correr el programa completo, para ello tendrá que dar clic con botón derecho sobre el archivo MainRLClass.java. Dar clic sobre Run As e inmediatamente Run as Java Application.

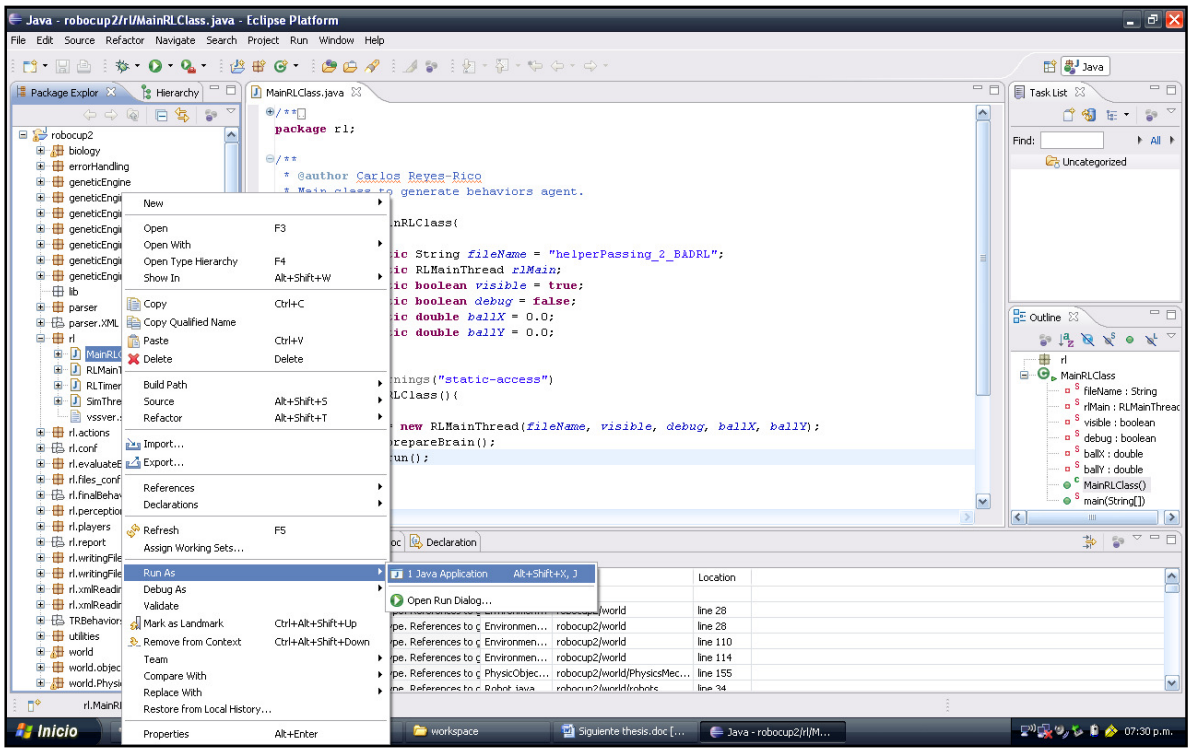


Figura 73. Ejemplo gráfico para correr el programa BADRL en Eclipse.

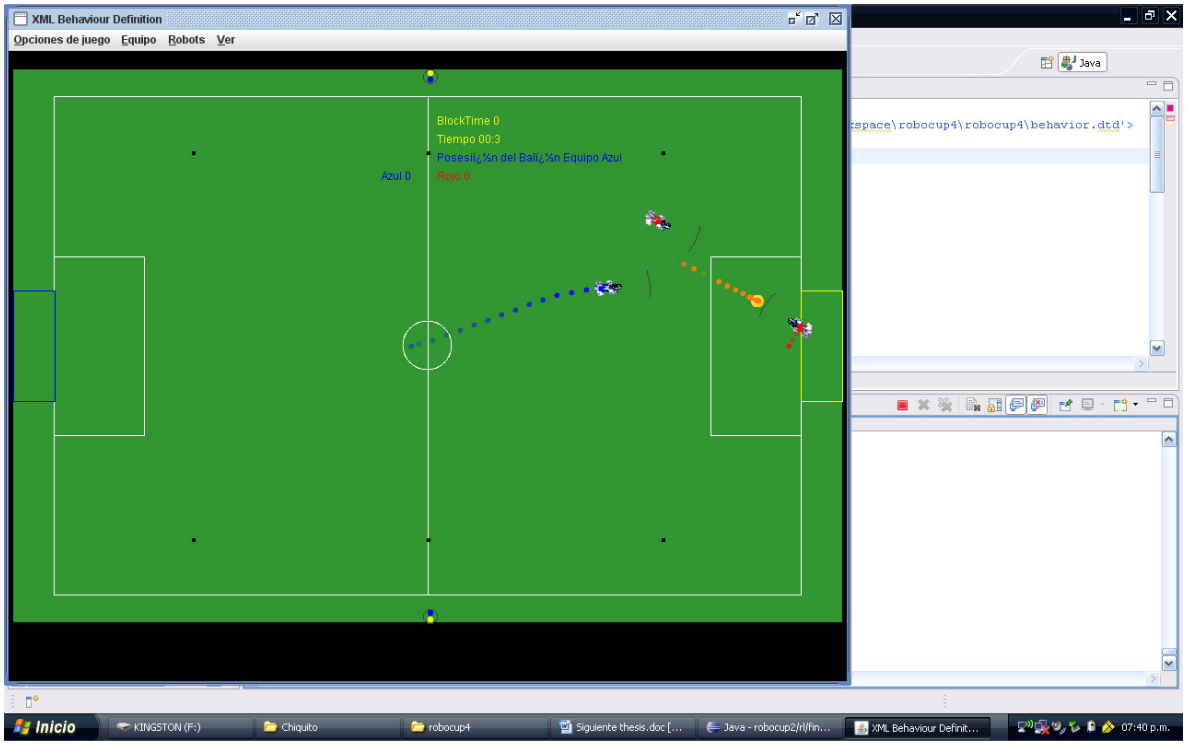


Figura 74. Corriendo BADRL en el simulador XML Behaviour Control.

7. MANUAL DE USUARIO.

A. CREAR AMBIENTE DE PRUEBAS

El ambiente es definido como el campo o lugar en el cual se encuentra inscrito un agente. Para que exista un modelo es necesario que el agente pueda tomar decisiones y actuar sobre un modelo. Además se obliga a que el agente tenga la capacidad de percibir los cambios que se efectuaron sobre un modelo que es capaz de modificar su estructura de acuerdo a las acciones de dicho agente. Cada vez que existan nuevos cambios sobre ésta, los agentes deben percibir esos cambios de acuerdo a los eventos que realiza el agente en el ambiente.

En este punto es importante señalar que cada comportamiento deseado requiere de un ambiente personalizado para su correcta evaluación y aseguramiento de un buen aprendizaje de una actividad en particular del agente.

BADRL utiliza el formato del ambiente que lo mande a llamar, en este caso, se programó que generara una copia del ambiente del *XML Behaviour Control* llamado *environment.xml* el cual tiene que ser modificado por el usuario cada vez que quiera cambiar de escenario para la evaluación de sus comportamientos [Figura 77].

```
<environment virtual_space="PlayGround2007" robot_delay="40"
      object_delay="150" friction="120">

      <!-- Approach Players -->
      <robot type="aibo" file="rl/finalBehavior/manualApproaching.xml"
            team="BLUE"       name="CuauBADRL"  initial_positionX="-400"
            initial_positionY="200"       position="LEFTATACKER"/>

      <!--  The others -->
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Seis"   initial_positionX="20"  initial_positionY="-40"
            position="DINAMICATACKER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Cinco" initial_positionX="20"   initial_positionY="40"
            position="DINAMICDEFENDER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Cuatro" initial_positionX="-20" initial_positionY="0"
            position="CENTERATACKER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Tres"  initial_positionX="-20"  initial_positionY="40"
            position="RIGHTDEFENDER"/>
      <robot type="aibo" file="TRBehaviors/doNothing.xml" team="RED"
            name="Dos"   initial_positionX="-20"  initial_positionY="-40"
            position="LEFTDEFENDER"/>
      <robot type="aibo" file="keeper2.xml" team="RED"
            name="Tala"  initial_positionX="200"  initial_positionY="0"
            position="DINAMICKEEPER"/>
</environment>
```

Figura 75. Ambiente especializado para evaluación correcta del comportamiento approachToBall.

B. ANALIZAR EL COMPORTAMIENTO DESEADO.

En el momento en que se tenga el ambiente de pruebas listo para correr pruebas, el siguiente paso es analizar el comportamiento que se requiere, esto es, el usuario debe saber el tipo de comportamiento que el agente debe aprender. El usuario debe saber las posibles acciones que puede utilizar el agente para la generación del comportamiento.

Es importante señalar en este punto que el usuario tiene dos opciones viables para obtener esta información. La primera de ellas es conocer el pool de acciones mínimas que el agente debe realizar para realizar el comportamiento. La segunda de ellas es integrar y usar todas las posibles acciones que puede realizar el agente.

Con lo anterior, ahora el usuario necesitará saber cómo evaluar el ambiente de acuerdo a las acciones que puede realizar el agente. Para ello, tendrá que analizar las posibles combinaciones de las acciones representativas por comportamiento y generalizar dichos casos. Esto servirá para que el usuario pueda encontrar de una manera más sencilla y rápida las reglas de definición e indicar si el agente esta en el camino correcto de aprender su objetivo o no.

C. PROGRAMAR POSIBLES COMPORTAMIENTOS.

BADRL es un sistema capaz de generar comportamientos colaborativos. El usuario tiene la capacidad de generar comportamientos colaborativos individuales, por lo que es necesario que éste implemente los comportamientos que van a colaborar con aquél que aprenderá.

Un ejemplo de lo anterior se puede visualizar de mejor manera como sigue. Si el usuario necesita que un agente aprenda a realizar una jugada conocida de pared, esto es, el agente mande un pase a otro agente compañero y éste le regrese el balón, entonces podrá realizar dos distintas maneras de implementar al agente compañero:

1. Generar manualmente el comportamiento del agente compañero.
2. Utilizar BADRL para que el agente compañero aprenda a realizar la obra de manera individual y el resultado arrojado, colocarlo como comportamiento en el ambiente de pruebas con el agente que ahora se quiere el aprendizaje.

En la figura 78 se puede observar un ejemplo de un comportamiento arrojado por BADRL como resultado de aprender a recibir la pelota y mandar un pase a un cierto ángulo con la fuerza necesaria para llegar justo a un punto dado. De esta manera el comportamiento fue utilizado para generar el ambiente de pruebas para que otro agente aprendiera el comportamiento de triangular y disparar a portería.

```
- <behavior name="passHelper" isInitialBehavior="true">
  - <state name="firstState" iterate="true">
      <action id="MOTION-doNothing" />
    - <transition state="voltea_izquierda">
        <event name="VISION-ballFound" operator="not" />
      </transition>
    - <transition state="tira_derecha">
        <event name="VISION-shootDistance" />
      </transition>
    - <transition state="voltea_izquierda">
        <event name="VISION-ballAtLeft" />
      </transition>
    - <transition state="voltea_derecha">
        <event name="VISION-ballAtRight" />
      </transition>
    </state>
```

Figura 76. Comportamiento passHelper, recibe la pelota y es capaz de arrojarla a un punto aproximado.

D. FASE DE PRUEBAS

1. *Generar pool de acciones posibles que puede realizar el agente*

Como primer bloque para el uso de BADRL, será necesario que el promador conozca todas las acciones posibles que puede realizar el agente. Un ejemplo de ello, es en el simulador XML Behaviour Control. El agente tiene disponibles 42 acciones entre las que se encuentran:

- MOTION-doNothing: Esta acción permite que el agente se quede estático sin realizar ningun movimiento.
- MOTION-turnRight: Esta acción permite que el agente pueda voltearse hacia un ángulo específico por la derecha.
- MOTION-goForward: Esta acción permite que el agente camine hacia delante con una velocidad dada.
- MOTION-shootLeft: Esta acción permite que el agente pueda disparar a portería siempre y cuando tenga en su posición el esférico.
- MOTION-lateralRight: Esta acción permite que el agente se pueda desplazar a la izquierda con una velocidad dada.
- MOTION-turnHeadRight: Esta acción permite que el agente pueda mover su cabeza hacia la derecha con un ángulo dado.
- MOTION-stopBall. Esta acción permite que el agente detenga el flujo de la pelota si éste se encuentra en su trayectoria de desplazamiento.

- MOTION-leftBlock. Esta acción permite que el agente pueda moversa hacia la izquierda tratando de detener el esférico. Generalmente esta acción es utilizada por los porteros para bloquear los disparos de los atacantes a la portería.

- MOTION-goToPosition. Esta acción permite que los agentes se ubiquen a una posición estratégica definida anteriormente por el programador de acuerdo al rol que manejan en el campo de juego.

- VISION-ballFound. Esta acción regresa verdadero o falso si el agente fue capaz de encontrar la bola.

- VISION-ballAtLeft. Esta acción regresa verdadero o falso si el agente observa la bola a su izquierda del eje principal.

- VISION-ballVeryNear. Esta acción regresa verdadero o falso si el agente observa la bola muy cerca de sí.

- VISION-goalVisible: Esta acción regresa verdadero o falso si el agente es capaz de observar la portería contraria de acuerdo a su posición actual.

- VISION-goalAtRight: Esta acción regresa verdadero o falso si el agente observa la portería contraria a su derecha

- VISION-positionAtFront: Esta acción regresa verdadero o falso si el agente se encuentra en la zona contraria a su portería.

- VISION-robotOnOwnPenaltyArea: Esta acción regresa verdadero o falso si el agente reconoce si está ubicado adentro de su área de penal.

Una vez que el programador reconozca estas acciones, tendrá que generar el archivo *actions.xml* con aquéllas que desea que utilice el agente. De hecho, si el programador sabe aproximadamente cuantas usará el agente, puede sólo escribir éstas, de lo contrario, tendrá que colocar todas las acciones posibles.

Para lo anterior, tendrá que generar una etiqueta llamada <*ql-action*> en la que colocará en primer lugar el nombre de la acción (texto libre para el programador) inmediatamente después colocará el atributo *isText* que define si lo que está adentro de la etiqueta <*ql-action*> es texto. En el caso concreto del simulador, siempre será verdadero. Cuando BADRL sea llevado a otro agente físico, tendrá que colocarse como falso para que se lea lo interpretado en la etiqueta com o acción directa que puede realizar el agente. Inmediatamente después tendrá el atributo *iterate* el cual nos indica si la acción durante su ejecución se estará repitiendo hasta un cierto tiempo y finalmente *action* y *params* los cuales nos indican que acción del simulador se realizará y con qué parámetros se efectuará dicha acción. Por ejemplo, si se requiere la acción caminar lento, entonces se manda a llamar la acción del simulador llamada *MOTION-goForward* con un parámetro de calibración 1. Si se requiere una acción de caminar rápido entonces se manda a llamar la acción del simulador llamada *MOTION-goForward* con un parámetro de calibración 4.

Finalmente adentro de la etiqueta <*ql-action*>tendremos los eventos que necesitan pasar para que se dispare dicha acción, para ello se cuenta con las subetiquetas internas llamadas <*ql-*

transition>los cuales tienen como atributos: el nombre que le quiere dar el programador a dicha transición y como texto adentro de esta etiqueta, la transición tal cual y como aparece en el simulador. Las transiciones dependen de la estructura interna de dónde se está desenvolviendo BADRL, en un simulador, es información que dá el mundo, si es en un agente, será lo que reconozca el agente en un momento determinado.

En la figura 79 se puede observar un ejemplo de acciones posibles que puede realizar el agente por medio del archivo *actions.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <qlearning-actionSet name="Acciones para JLV" version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="actionSchema.xsd">
    <ql-preconditions language="xml" action="action" transition="transition" numberOfActionsSelectedPerBehavior="4" />
    <!--  approaching BADRL  -->
    <ql-action path="rl/finalBehavior/" name="approachingWithAligningsBADRL" isText="true" isBehavior="true" />
    <!--  Primer tiro  -->
  - <ql-action name="tira_izquierda_lejos" isText="true" iterate="false" action="id='MOTION-shootFront' param1='700' param2='30'">
      <ql-transition name="event">name="VISION-shootDistance",</ql-transition>
    </ql-action>
    <!--  Tiros a porteria  -->
  - <ql-action name="tira_izquierda_porteria" isText="true" iterate="false" action="id='MOTION-shootFront' param1='700' param2='30'">
      <ql-transition name="event">name="VISION-goalAtLeft",</ql-transition>
      <ql-transition name="event">name="VISION-shootDistance",</ql-transition>
    </ql-action>
  - <ql-action name="tira_derecha_porteria" isText="true" iterate="false" action="id='MOTION-shootFront' param1='700' param2='-30'">
      <ql-transition name="event">name="VISION-goalAtRight",</ql-transition>
      <ql-transition name="event">name="VISION-shootDistance",</ql-transition>
    </ql-action>
  </qlearning-actionSet>
```

Figura 77. Conjunto de acciones posibles del agente por medio del archivo actions.xml.

2. *Generar reglas para evaluación*

Como fue explicado anteriormente, este archivo es analizado de manera secuencial y se terminará de revisar hasta que una de las reglas cumpla con las características. Por lo que es necesario que el programador genere sus reglas primero por aparte y después pasarlas al formato especial para el archivo *rules.xml.* Para generar las reglas es necesario conocer de fondo cuáles son los eventos que tienen que suceder en el ambiente para que pueda dispararse y que el agente pueda reconocer si va bien o mal en su etapa de aprendizaje.

Para generar las reglas será necesario generar una etiqueta llamada <*ql-rule*> con los siguientes atributos:

1. id = es el número identificador de la regla. Tendrá que ser consecutivo para cada una de las reglas.
2. name = el nombre asignado por el programador para cada regla, para reconocer qué realiza dicha regla.
3. useTimer = el tiempo necesario que tendrá que pasar para que se dispare la regla

Adentro de la etiqueta <*ql-rule*>se tendran 3 etiquetas principales:

1. <*ql-event*> = señala el evento que tiene que ocurrir para que se dispare la regla. Pueden ser varios enventos por lo que el programador puede utilizar las etiquetas <*and*> o <*or*>
2. <*reward*>= señala la recompensa que el agente obtendrá si se ejecuta dicha regla. En este caso se le menciona mediante dos atributos *goWell* y *goBad* con parámetros de verdadero o falso.
3. <*debug*>= Información que necesita el programador (texto libre) si requiere información adicional en el reporte que se genera, si se dispara dicha regla. Por ejemplo, si el

programador necesita saber la posición final que tuvo el agente al final de la evaluación de la regla, entonces podrá colocar *positionX* en el atributo info y así podrá ver la información de la coordenada X donde estuvo localizado el agente.

En la figura 80 se puede observar un ejemplo de reglas posibles que puede realizar el agente por medio del archivo *rules.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <qlearning-ruleSet name="Reglas de evaluacion para dribbiling" version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="ruleSchema.xsd">
   <!--   SIEMPRE CHECA MAYOR O IGUAL A USETIMER 7000 - 10 SEC.   -->
   <!--   TARGET   -->
  - <ql-rule id="1" name="OBJECTIVE_TARGET" useTimer="7000">
     <ql-event name="goal" value="VISION-shootDistance" function="isGoal()" />
     <reward goWell="true" goBad="false" />
     <debug text="Target Reached!! GOOOAL!!" info="positionX" />
   </ql-rule>
   <!--   Position greater than 0   -->
  - <ql-rule id="2" name="Position greater and ball found" useTimer="7000">
   - <and>
       <ql-event name="agent" value="positionX" function="greaterThan(100)" />
       <ql-event name="eventBF" value="VISION-ballFound" function="isActive()" />
     </and>
     <reward goWell="true" goBad="false" />
     <debug text="Position =" info="positionX" />
     <debug text="and Ball Found" />
   </ql rule>
  - <ql-rule id="3" name="Position greater and ball not found" useTimer="7000">
   - <and>
       <ql-event name="agent" value="positionX" function="greaterThan(100)" />
       <ql-event name="eventBF" value="VISION-ballFound" function="isActive()" operator="not" />
     </and>
     <reward goWell="true" goBad="false" />
     <debug text="Position =" info="positionX" />
     <debug text="and Ball Found" />
   </ql-rule>
```

Figura 78. Conjunto de reglas posibles para evaluar a un agente para un comportamiento específico.

3. *Probar acciones y reglas corriendo Q-Learning*

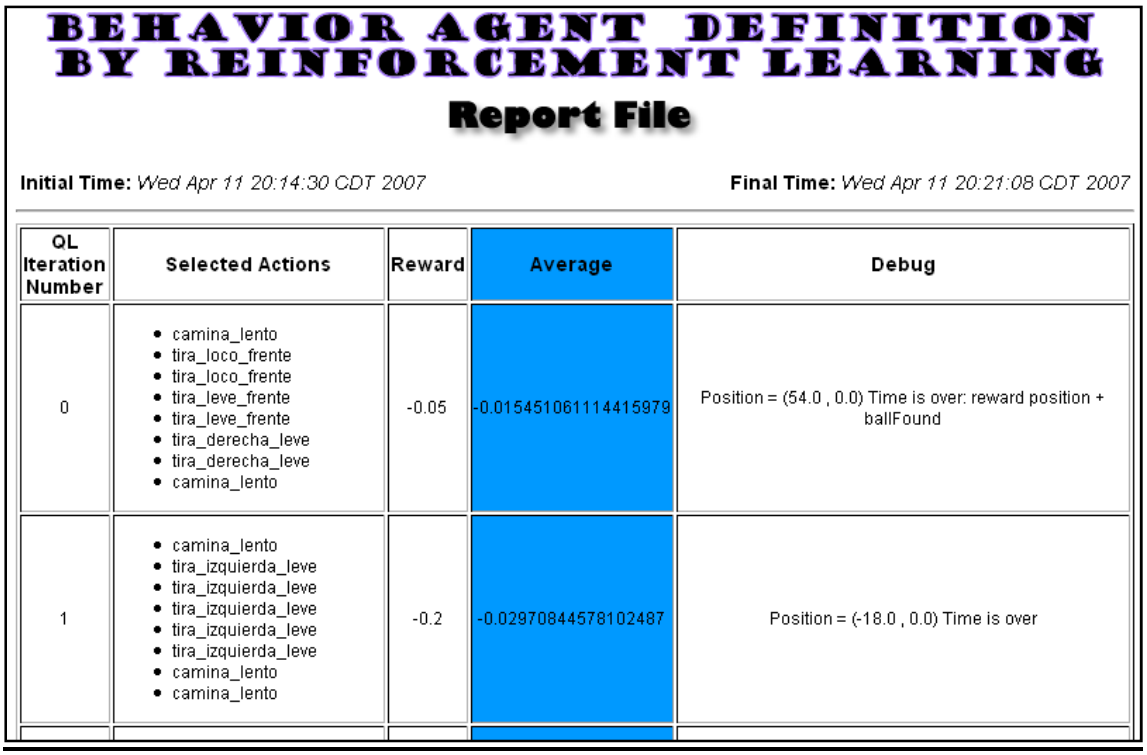
Una vez que ya se hayan colocado y generado los archivos *rules.xml* y *actions.xml* el programador podrá correr el programa del simulador junto con BADRL y revisar a detalle lo que está ocurriendo durante la simulación. Para ello la clase principal MainRLClass.java contiene atributos modificables para que el programador pueda revisar el avance específico del simulador. Por ejemplo el atributo *visible* el cual permite ver gráficamente lo sucedido en el aprendizaje.

4. *Analizar reporte final*

Ya que el programador revisó que BADRL terminó su funcionamiento, podrá revisar el reporte generado automáticamente en la carpeta *rl/report* en donde podrá apreciar qué realizó el sistema en cada iteración, cuántas iteraciones realizó BADRL, el tiempo de inicio y fin del programa, las acciones utilizadas en cada iteración, la recompensa y la regla arrojada en cada iteración y el promedio que tiene la tabla de los Q-values del algoritmo Q-Learning. En la figura 81 se puede observar un ejemplo del reporte final que arroja el sistema BADRL.

5. *Obtener comportamiento final aprendido*

El comportamiento final que arroja BADRL se encuentra en la carpeta *rl/finalBehavior* con el nombre que puso el programador en el clase principal del simulador MainRLClass.java. El nombre se puede cambiar en el atributo de esta clase llamado *fileName*



BEHAVIOR AGENT DEFINITION BY REINFORCEMENT LEARNING

Report File

Initial Time: *Wed Apr 11 20:14:30 CDT 2007* **Final Time:** *Wed Apr 11 20:21:08 CDT 2007*

QL Iteration Number	Selected Actions	Reward	Average	Debug
0	• camina_lento • tira_loco_frente • tira_loco_frente • tira_leve_frente • tira_leve_frente • tira_derecha_leve • tira_derecha_leve • camina_lento	-0.05	-0.015451061114415979	Position = (54.0 , 0.0) Time is over: reward position + ballFound
1	• camina_lento • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • tira_izquierda_leve • camina_lento • camina_lento	-0.2	-0.02970844578102487	Position = (-18.0 , 0.0) Time is over

Figura 79. Ejemplo visual del reporte final que arroja BADRL después de aprender un comportamiento

.

Fase de investigación

6. *Cambiar parámetros de configuración para el uso de Q-Learning*

Una vez, que el programador tenga como resultados comportamientos y reportes con cierto grado de satisfacción en donde pueda deducir si se está logrando el objetivo o no, viene la parte de la fase de investigación, la cual viene enfocada a modificar directamente los parámetros y forma de recompensar del algoritmo Q-Learning. Para ello, BADRL consta del archivo *conf.xml* que contiene por default los parámetros que hacen funcionar al algoritmo para generar comportamientos, sin embargo, si el programador necesita realizar análsis e investigar qué pasa con nuevos comportamientos con otros parámetros, por lo tanto podrá tener acceso a este archivo y cambiarlos a su gusto. En la figura 82 se puede observar el archivo *conf.xml* con los parámetros de configuración de Q-Learning.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <qlearning-app xmlns="http://tempuri.org/conf2.xsd" name="XML Behavior Control with BADRL" version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="configurationSchema.xsd">
  - <parameters>
      <alfa-param value="0.5" />
      <gamma-param value="1" />
      <lambda-param value="0.8" />
      <boltzmann-param value="false" />
      <temperature-param value="1" />
    </parameters>
    <!--   qLearning rewards   -->
  - <rewards>
      <movingWell-reward value="0.1" />
      <collides-reward value="-0.05" />
      <default-reward value="-0.01" />
    </rewards>
    <!--   qLearning actions   -->
    <ql-actions file="actions.xml" />
</qlearning-app>
```

Figura 80. Archivo conf.xml que sirve para modificar los parámetros de configuración de Q-learning de BADRL

7. *Volver a la fase de pruebas*

Finalmente, ya que el programador genere los cambios respectivos en este archivo, podrá regresar a la fase de pruebas para cambiar acciones o reglas para los agentes y volver a correr el sistema BADRL.