**Game Project Report**

Ludo

Submitted By

Praful Bhosale

Net ID: pbb140230

CS 6364.002 - Artificial Intelligence

University Of Texas, Dallas

Introduction

This project report describes the process involved in developing Artificial Intelligence for 2-D computer board game known as “Ludo” using Greenfoot java IDE.

Greenfoot allows object orientation using Java. Create 'actors' which live in 'world' to build simulations, and other graphical programs. Greenfoot provides visual and interactive environment to implement various AI concepts.

This report discusses the game overview, including the description and game play, and then focuses on the game design, approach describing how the game is implemented and the programming functions, algorithms and libraries used in the design.

Description

I developed board game named Ludo which can be played between four players, in which the players race their four tokens from start to finish according to die rolls. Each player gets a turn to roll dice alternatively. The dice shows a number from 1 to 6. There is finish place (Goal) for each player at the center shown with circle ‘G’. All four players have their houses shown with different color at the four corners of a board. Each player has 4 tokens in their house which take part in a race.

At the initial state of the game, each player's tokens are out of game track and staged in one of the large corner areas of the board in the player's color (called the player's house). When able to, the players will enter their tokens one per time on their respective starting circles, and proceed to race them clockwise around the board along the game track (the path of circles not part of any player's home column). When reaching the circle below his home column, a player continues by racing tokens up the column to the finishing goal circle.

Player 1: Red Color with four tokens, Red colored House.

Player 2: Green Color with four tokens, Green colored House.

Player 3: Blue Color with four tokens, Blue colored House.

Player 4: Yellow Color with four tokens, Yellow colored House.

There are some rules that each player needs to follow while playing.

Rule 1: Players can’t start a race for their respective colored token unless player gets 6 on dice (note: this applies to each of the four tokens).

Rule 2: Player’s token will be killed if some other player’s token come to same place. Then killed token will send back to its respective house location.

My Approach

I started by planning the class hierarchy of a game, and constructing a UML of the class inheritance on paper. Then I researched on different game development environment softwares and decided to go with Greenfoot java IDE which is suitable for my further game plan. Then I prepared rough class diagram showing all classes with variables & member functions. This organized my plan for algorithm creation, coding and setting up the class hierarchies. Class hierarchy is shown below in Fig.1 at right side.



Fig. 1: Class Hierarchy of Ludo Game

**Inheritance Design:**

**greenfoot   
Class World**

java.lang.Object

extended by **greenfoot.World**

World is the world that Actors live in. It is a two-dimensional grid of cells. All Actor are associated with a World and can get access to the world object. The size of cells can be specified at world creation time, and is constant after creation. I created my own world object named ‘Board’ extending this world class with fixed dimensions. I have set the background image to this world class at the time of creation.

**greenfoot   
Class Actor**

java.lang.Object

extended by **greenfoot.Actor**

An Actor is an object that exists in the Greenfoot world. Every Actor has a location in the world, and an appearance (i.e. an image icon). Every object that is intended to appear in the world must extend Actor. Subclasses can then define their own appearance and behavior. I created my own actor classes like Player, DiceTurn, RollDiceAnimation, PlayGame, PlayerSelection, Scoreboard, Ruleboard, Text, StartUpMenu and Gameover. Player class is further extended by RedPlayer, BluePlayer, YellowPlayer & GreenPlayer classes. RollDiceAnimation class is further extended by Dice class which is further derived by DiceScore class. Each of these class represents different actors living in a world with distinct image icons.

**Implementation Details:**

* Initializing Players:

Once game starts each player token gets initialized with different variables which defines the Initial state of each Player. According to number of players selection from Game Menu. Board initialized with human players & computer players (i.e. AI agent) on board.

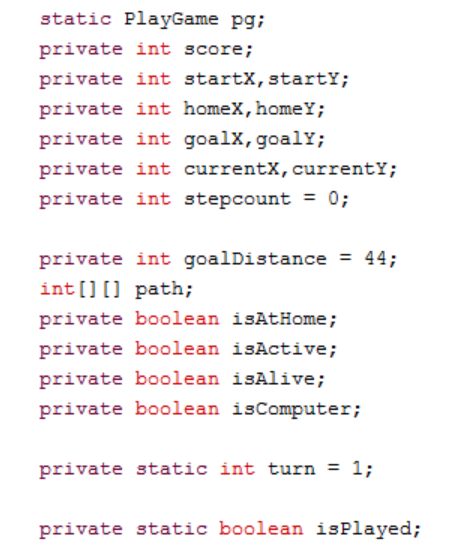


Fig.2 Initializing Variables to define different states of Player token

* Defining Heuristic for Game:

I implemented heuristic search concept of Artificial Intelligence to select best move for AI agent (Computer player). I initialized each player token with path from its respective start circle to goal circle using integer 2-D array named Path which stores X & Y coordinates of each step (i.e. path circle). Each circle along the path represents step. Token has to race across 44 steps (i.e. path circle) to finish the race. Fig.2 describes heuristic defined for red player & its data. Each computer player uses modified best first search algorithm to drive search with this given heuristics & makes the game planning process efficiently.

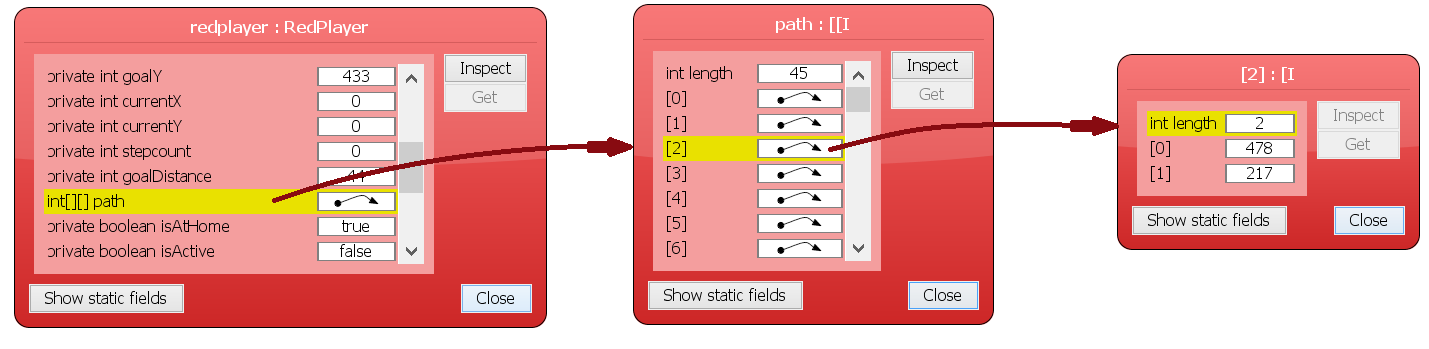


Fig. 3: Heuristic Function for setting Path

* Game Menu Implementation:



StartUpManu class adds menu on board with player selection options. Once you select number of players from menu & click play. PlayGame class sets variable NoOfPlayers = selected value. This value defines no of human players in a game. For e.g. if we select ‘3’ from menu, so there will be 3 human players & 1 computer player.

Fig.4: Game Menu

* Rolling Dice Implementation:

RollDiceAnimation class displays throwing dice animation which triggers dice class to call method RollDice(). The RollDice() method returns dice score for a player & DiceScore image object gets added on scoreboard for that particular score with relevant notification.

* Blackboard Implementation:



As we click ‘Play’ button on game menu, Blackboard actor gets added onto board. It shows player turn & its score (i.e. No of token that reached goal). It also sets 4 players on board with respective image icons for Human player & Computer player based on NoOfPlayers. Text class displays changing score. DiceTurn class updates player Fig.5 Blackboard turn as game proceeds.

* Scoreboard Implementation:

Score Board actor added onto board once game starts. As player rolls dice random score gets generated & it gets displayed on scoreboard. Notification class adds different notification objects according to game state which directs human player to proceed.

Fig.6 Scoreboard

* Sound implementation.

Greenfoot IDE has in-build method named playSound, which takes sound file as an input. The following formats are supported: AIFF, AU and WAV. I have included different sound effects at various stages in my game at the start, when any player gets killed, reaches goal or wins the race.

**Challenges & Arguments during implementation:**

* To synchronize Player turn, Player movement & respective notification on scoreboard:

After implementing main algorithm for player movement, I faced some issues while synchronizing player turn & respective notification board. So I put checks on static boolean variable named Played. Once player plays its move makes Played <- true which sets turn for next player & if player is human then it will give notification regarding player selection.

* To display dice turn actor between remaining players after we have our 1st winner, or 2nd winner.

Once we have our 1st or 2nd winner, tokensReachedGoal will be 4. While setting turn I am checking

Static variable tokensReachedGoal of remaining player to set next turn.

Algorithms

I have implemented Best-first search algorithm which explores best player token among active pieces for each player according to a specified rule. I modified best first search to use my heuristic function. Below I described the algorithm used for Computer Player Movement.

* Computer Player Movement:

Function movePlayer (Player.class, int score)

Input: 1) Player.class , who has a turn to play.

2) score, which displayed on dice.

List<Player> p <- Store all objects of input player class.

// Check if any Player token reaches Goal

For each x in P

Do

Check if x isAlive()

Do check if x isActive()

{ Do

futureStepCount <- currentStepCount + score

Check if futureStepCount < 45 as goal distance for each player is 44

{Do

Check if x reaches Goal

{

Do

Set x isAlive() <- false.

Check if score == 4 if Yes then set winnercount according other player.

Set Played <- true

Increase tokensReachedGoal by 1

}

}

}

End For loop

// if still Player is not played then checking if it kills other player token

Check if Played is false

{

For each x in P

Start loop

Do

Check if x isAlive()

Do check if x isActive()

{Do

futureStepCount <- currentStepCount + score

Check if futureStepCount < 45 as goal distance for each player is 44

{Do

List<Player> d <- stores all objects at path[futureStepCount] – x & y

Coordinates

While d has object

{

Do

Set Player p <- d.next()

Now kill p & set x at this location

Set p isActive() <- false

Set Played <- true

}

}

}

End For loop

}

// if score is 6 & Player is still not played then making other player token active

Check if Played is false

{

For each x in P

Start loop

Do

Check if x isAlive()

Do check if x isActive() & score = 6

{Do

List<Player> d <- stores all objects at start position of player x

While d has object

{

Do

Set Player p <- d.next()

Check if p != x

{Do

Now kill p & set x at this location

Set p isActive() <- false

Set Played <- true

}

Else

{Do

Set x at start location

Set Played <- true

}

}

}

}

End For loop

}

End Function

**Lesson learned from this Project:**

* Problem formulation is important step in AI based games. Problem Formulation sets the plot for designing, algorithm analysis & coding.
* Implementation of few AI concepts in game environment with efficiency.
* While searching for ideal platform for my game, I came across many game development IDE’s and their significance in various game scenarios.
* You need to deal with different problems at all stages of development. It is good to write every problems down & try to find best logical solution.
* Music and sound are half of your game. Many times developers leave them for last, and games suffer for it. A game that doesn't react aurally to the player's actions feels dead.
* Simple ideas can be very compelling.