Version	Name	Comment	Date
1.0	Martti Vasar	Initial text and documentation	25. November 2010
1.1	Martti Vasar	Fujaba drawings	30. November 2010

Project Report
Systems Modeling, Fall 2010

Team members: Taavet Loogväli, Mihkel Jõhvik, Allan Reinhold, Martti Vasar
Team member codes: 97854, A62129, A72091, B04849

Task

Design and implement the game Mancala (AI and network support optional). Use the objects first design method. This means, start with scenarios, derive user stories, create object diagrams, sort into usecases, derive class diagrams, use object game or note-pad test to define methods, implement methods. Iterate, refactor, use/apply design patterns, refine documents form the previous steps. Implement at least a hot seat version of Mancala (using two players). Implement a history function for keeping scores of specific matches.

Game

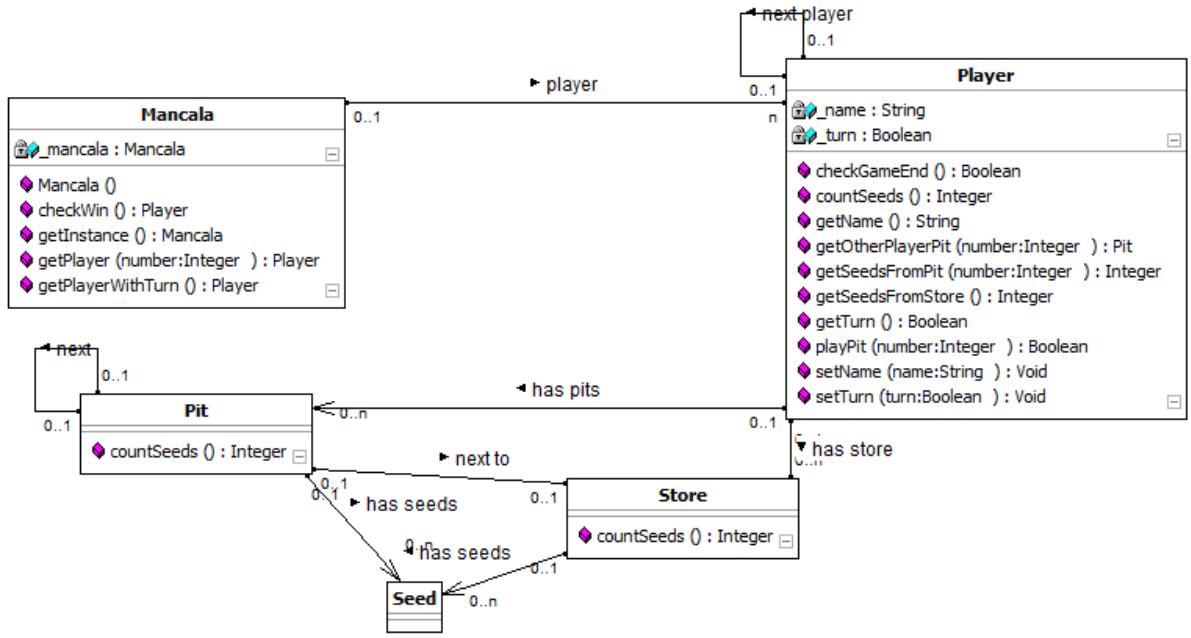
Mancala games share a common general gameplay sequence. Players begin by placing a certain number of seeds, prescribed by the variation in use, in each of the pits on the game board. A player may count their stones to plot the game. A turn consists of removing all seeds from a pit, sowing the seeds (placing one in each of the following pits in sequence), and capturing based on the state of board. This leads to the English phrase "Count and Capture" sometimes used to describe the gameplay. Although the details differ greatly, this general sequence applies to all games.

Source: http://en.wikipedia.org/wiki/Mancala

Desicions

As Mancala has many forms and rules, we first set up our Mancala game rules. Our decision was to design game with 4 seeds per pit. It is still possibile to change it from the source code, but we did not put an option to change it from inside the game application. We used MVC (Model-Controller-View) paradigma to make the game. Model part is done entirely in Fujaba and it includes function to get information about seeds, how many seeds are in one pit, how many seeds are in one store, whose turn it is, is there a game end condition. Controller and View classes are made by written the code.

Model



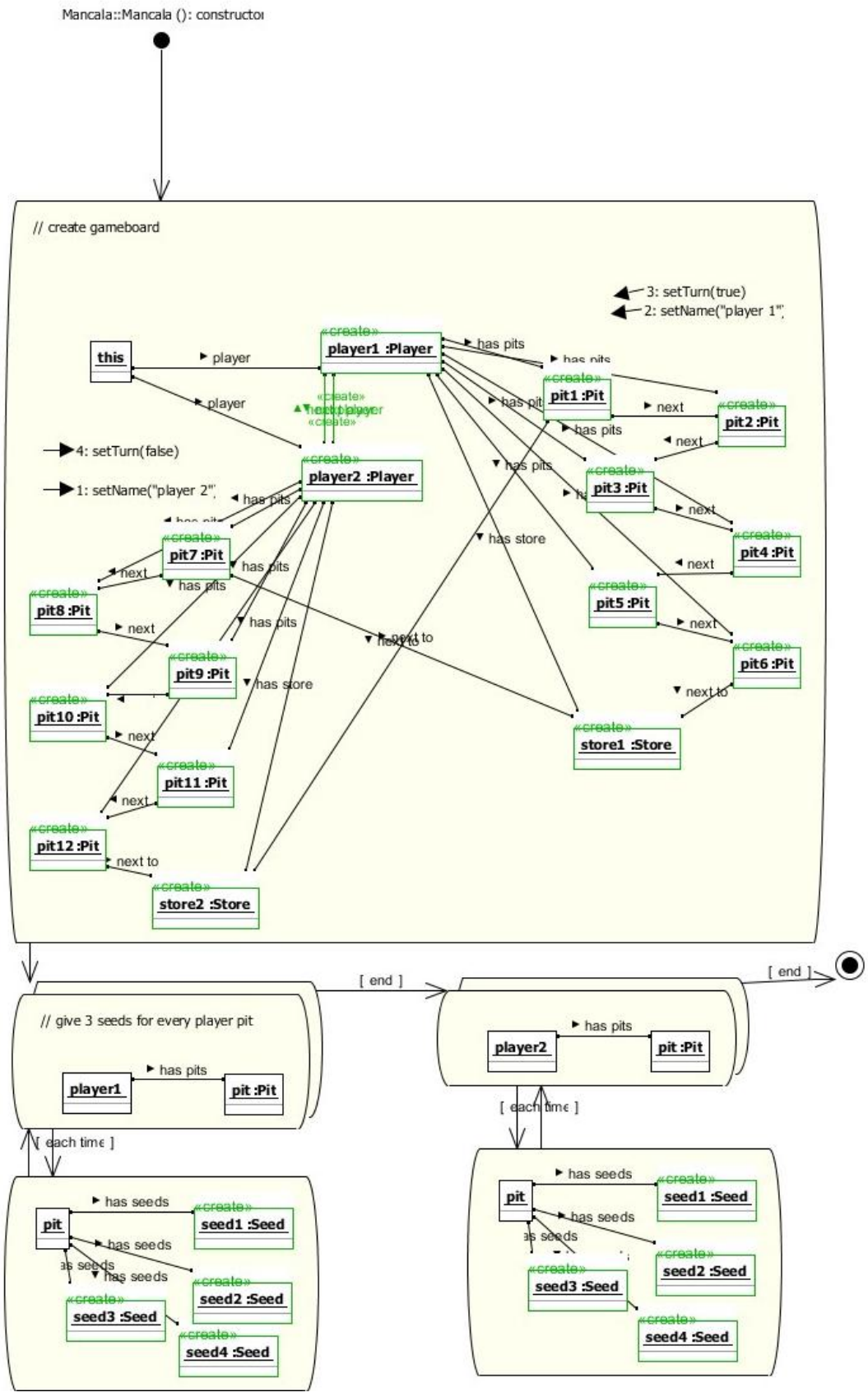
In fujaba model we have main class Mancala, which is named as *mancalaGame* in code and is part of MVC – it represents the model. We have functions to get player by numeric value (0 – first player, 1 – second player) or we can get player by turn, so it always returns the player with turn.

We have 2 players connected with Mancala class and players are connected with each other, so when one is making turn and the turn ends, the turn in normal condition is passed to the next player. Player class has methods to check, if current player meets the requirements for game end, this is when no more seeds are in current player's pits. We can count all the seeds from current player pit and store by method *countSeeds()*. Also we have some setter and getter functions for manipulating with name and turn. The most importan function here is *playPit()*, where current player plays given pit, takes all the seeds from that pit and sows them counter-clockwise to other's pits and stores. It returns true, when move is allowed and completed successfully. It returns false, when it is not current user's turn or user tries to take seeds from pit, where is no seeds.

One player has 6 connections with pit and one connection with store, pits are connected by each other as next relation and final pit (6th) is connected with store. This helps to model in Fujaba much easily the gameplay, when player takes seeds and starts to distribute them into next pits and stores. Also we have function countSeeds() for classes Pit and Store, which counts seeds from one's pit or store. For clarity purpose, we modeled Pit and Store classes seperately. Also class Seed is modeled out. It is just an empty class, but other way was to model it as integer value into class Pit or Store.

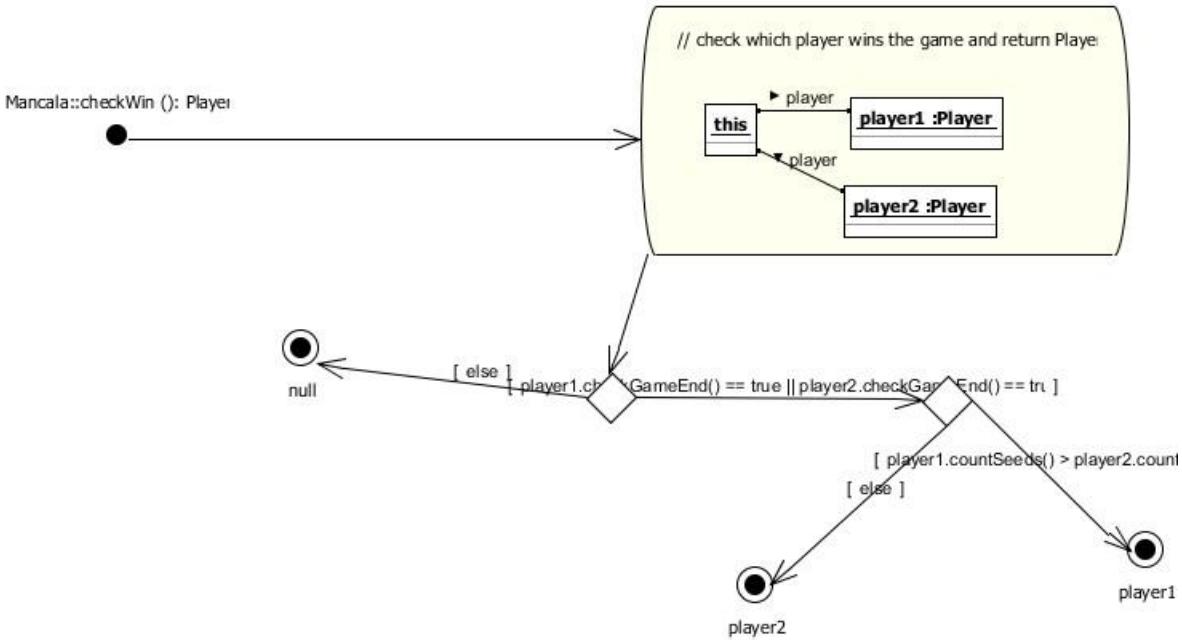
Model – Mancala class method Mancala()

This class generates gameboard with two players, two stores, 12 pits and 48 seeds. Each pit contains 4 seeds. First player with name „Player 1“ gets turn.



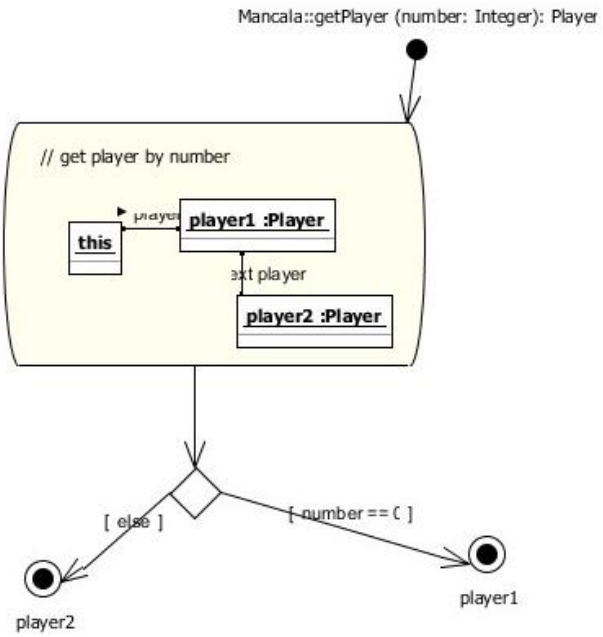
Model – Mancala class method checkWin()

This function returns player who won the game as Player class type. When we do not have game end condition, it returns null.



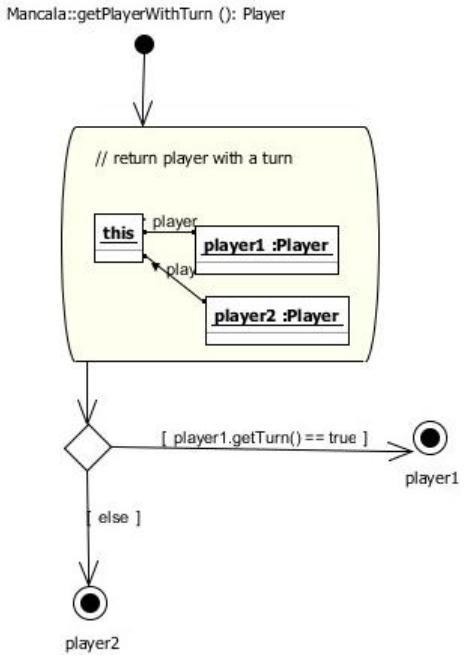
Model – Mancala class method getPlayer(int number)

This function returns Player class type. As we have two players defined, int number can be 0 or 1. For selecting first player, we have to define number as 0 and in othercase with number 1. Players are connected with each other by link next player.



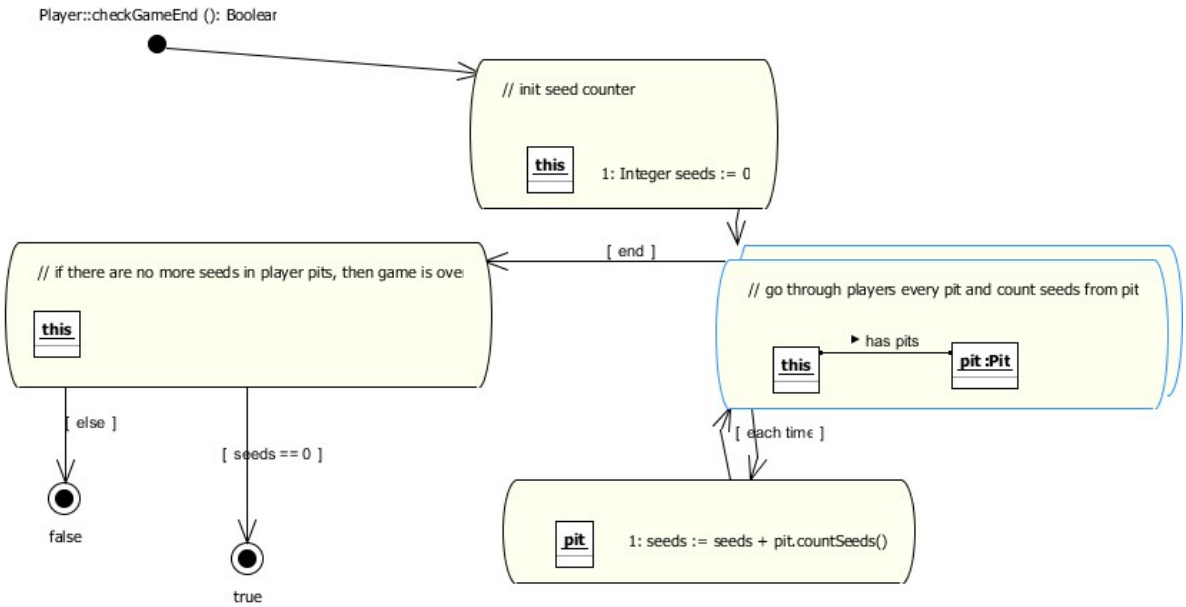
Model – Mancala class method getPlayerWithTurn()

This function returns Player class type user with turn. For implementing hot seat version, it is easy to just use getPlayerWithTurn() and then use playPit() as simulating the game progress.



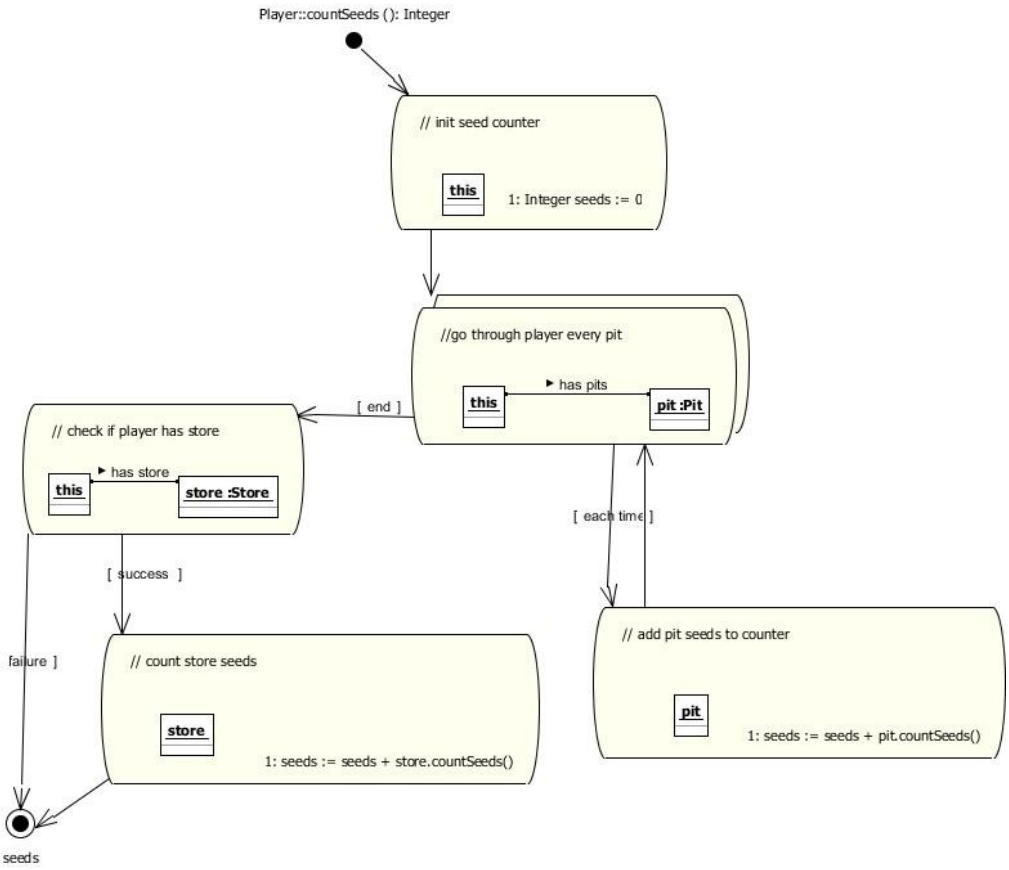
Model – Player class method checkGameEnd()

Returns true if selected player has end game condition, this means that he or she does not have any seeds left in the pit, otherwise it returns false. The fujaba model goes through every pit what player owns and checks, how many seeds are there. If total seeds count is 0, then return true, otherwise false.



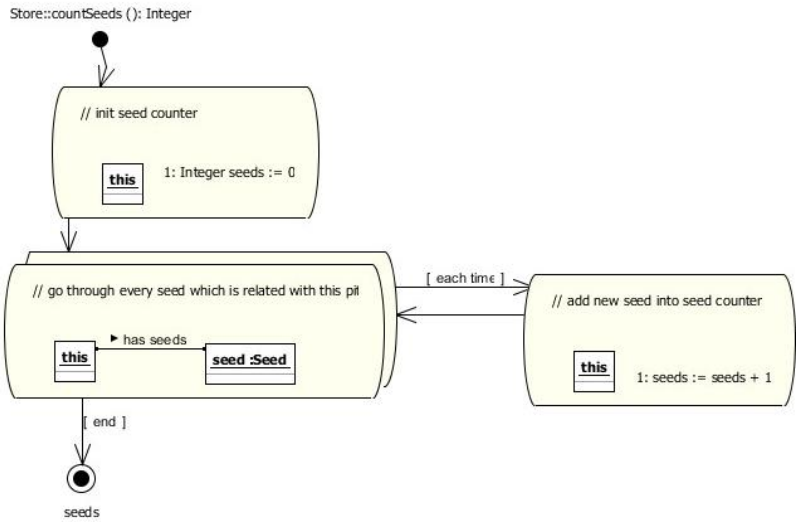
Model – Player class method countSeeds()

Returns overall seeds count from current player's pits and store. This is needed for checking who wins the game in the end.



Model – Pit and Store class method countSeeds()

Returns seeds count from selected pit or store.



Model – Player class method playPit()

Returns true, if move is valid and succesfully finished and returns false otherwise. This method lets selected player to try to sow seeds from one pit to other pits in counter-clockwise on the gameboard. As the graph is too big to show in this document, but it is possible to see from the Java project in the file Mancala.ctr. The main idea is to first check conditions, that is it currrent player's turn and if player selected valid pit (from 1 to 6, other numbers should be ignored). The next step is initilize some variables for helping purpose to find correct pit.

If the correct pit is found, seeds are counted and put into variable. For simplifying graph, plus one is added to the seeds count. This is done because in case, when we select last pit, we do not have next pit and so one we should have do some logical connection to the store. But as we sow the seeds first into the pits and then store, we could skip that simply by putting first seed into the same pit where we took it. We delete it later, so everything work correctly. If the seeds count is 0 (in our case 1, as we have added one more), then return false as we did not selected valid pit for sowing. Otherwise turn is given to other player and seeds are deleted from one pit.

First we put collected seeds to others pit, when pits are over, we take current player store. As the store ends we take other player and start to put seeds into their pit, if we still have seeds left and we continue this way until we run out of picked up seeds. Also we have to check conditions where last seed is put into current player store, this automatically means, that this player can take another turn. Also if the last seed lands into current player pit, where is no seeds and in the opposite direction other player's pit has at least 1 seed in, we can capture other player's pit seeds, the last seed and eventually put it into current player's store.

Implementation

Game uses JSwing component JFrame to show the content. As drawing pictures and lines onto java canvas, flickering can occure. Do avoid this, we set up double buffer, so when one of the frame is drawn, then it is shown to the user.

Clicking and mouse handling is done with MouseListener and MouseMotionListener. MouseMotionListener highlights pits, when mouse moves over them and MouseListener captures clicks and tries to figure out which pit or which menu element player just clicked.

Game can be played in hot seated mode, which is two players playing on single board and network mode, where one player is hosting and other player is joining with game. Hosting a game freezes up JFrame, this is an issue as the Socket waiting other player is thread blocking. The JSwing components return into its normal state when other player joins with server. One possibile way to avoid this thread blocking is to create seperate thread for waiting a player and implement timeout or user interruption.

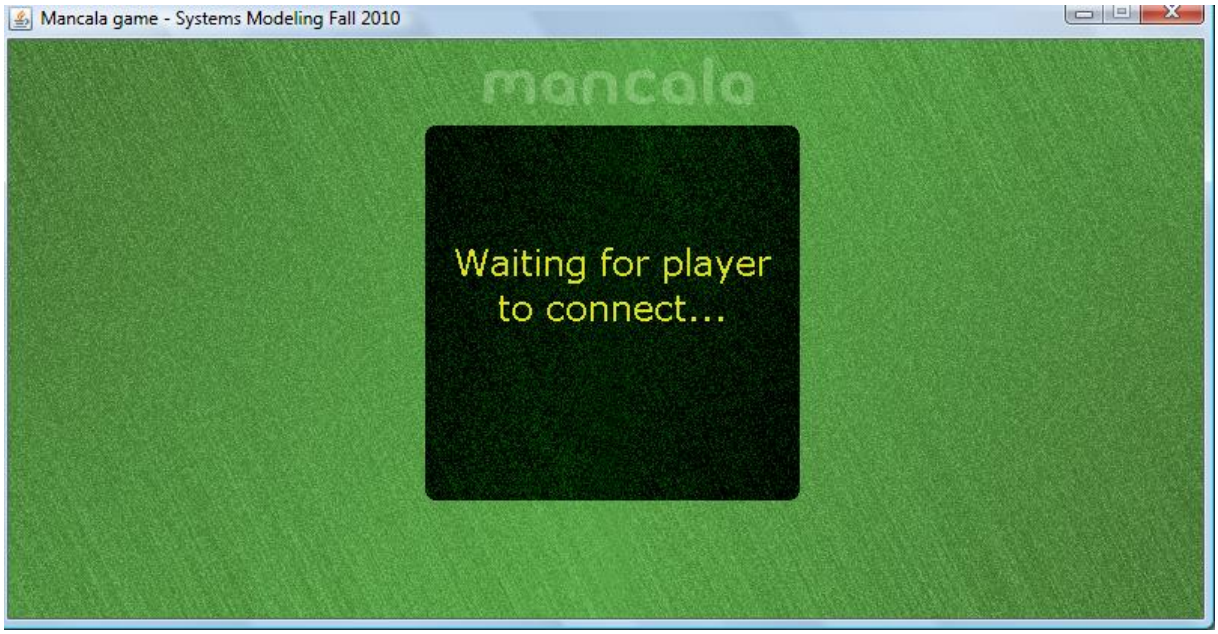
Network packets are simple. They just contain the number from 0-5, from which pit player takes seeds. It can be hacked, as it does not have any error check or crypt function to verify, is this packet correct and is it sent by game or it is generated by people. But for simple gaming it does its thing and works well.

Screen captures

Main screen, where user can select from different option

Hot seated game, where it is first player's turn and right now third pit is highlighted as the mouse is one the pit. Also current state of game is shown for player 2 in upper left corner and player 1 in lower right corner.

Winning screen, showing that player 1 just won the game.



Player is hosting a game and waiting for other player to connect

Network game showing that it is other player's turn.