Documentation for Mancala project

Introduction

This documentation describes the program Mancala developed as a project work for System Modelling. The game was developed using a system modelling tool plug-in for Eclipse called Fujaba. Some screenshots of the plug-in are included in this documentation.

Mancala

In this document Mancala is a *player vs. player* board in which players compete for getting seeds. Seeds are distributed in the beginning of the game to 2**n* houses. Player can choose a house to play each turn and the seeds in the house are distributed to the next house and possibly to the players store. The turn can stay for a player multiple turns, depending on how the seeds get distributed.
Exact rules for the game developed in this project are the following:

- the board has 12 houses. 6 on each side.
- player 1 plays the first move
- player can play one of his own houses that contain more than 0 seeds.
- turn is changed if the last seed to be distributed from the played house doesn't end in the players store.
- the player captures opponent seeds if his last seed played goes to an empty house and the opponent has seeds in the respective house. In this case the opponents respective house gets emptied to the players store and also the played seed is put into the player's store.
- game ends when the current player doesn't have any seeds in his houses.
  any of the players? -some versions have implemented it with the current player- rule
- in the end of the game all seeds of the players houses gets emptied to the players store.
  in the current implementation and the user stories we did not use this rule
- the winner is the player who has the most seeds in the end of the game.

The Mancala program

The program developed for this project is a java project created mainly with a graphical modelling tool plug-in Fujaba. The modelling involves creation of classes, associations between classes, attributes and methods describing the classes. In this section of the project documentation we will describe the classes we decided to implement and use.
We started the development of the program by writing 30 user stories that contains events that can take place when playing the game. Based on these user stories we draw object diagrams that describe these situations more graphically. The choice for classes to be modelled was based on nouns that occurred in the previously written user stories and object diagrams. These classes contained the house, store and the player and in the beginning also the seed.
The seed class was soon removed because it didn't end up containing enough information and

it made the object diagrams look messy. Modelling the seeds as attributes of houses felt like a more natural way to implement it and made the model easier to perceive. Also due to the the similarity of the house and the store, we combined these two so that the house implements a store.
In addition to the visible elements in the physical game we also modelled the classes application and turn. Turn was a also seen as a no information containing small element, but since it was a singleton and didn't mess the modelling view, we decided to implement it as a class. This way it is also much easier to perceive the turn changing. The application was a way to model things happening in the screen before the actual playing is started.

Class House

The House class contains the information about houses, their relations, their seed counts and the relations to the players. The house class implements the store with the added attribute index. The house class contains the methods and attributes

- getOpposite() : return the respective opposite house
- getPlayer() : return the owner of the house
- index : index of the house

Class Store

The store contains the count of seeds the player has captured.

- accept() : for visitor pattern
- seeds : count of seeds in the store

Class Player

Player class stores the players name, given before playing, and a first player-tag, which is set to 1 for the player who started the game.

- capture(i) : captures the seeds from house with the index = i
- checkEnd() : check if the player can play
- getHouse(i) : return the house with index = i
- getOpponent() : return opponent player
- initStoreAndHouse() : initialize the stores and the house
- playHouse(i) : playes the house with index = i
- name : name of the player
- firstPlayer : firstplayer tag

Class Turn

Doesn't contain any information but it's association to the players indicate the current turns player.

- instance : the instance of the turn as the singleton pattern requires

Class Application

The Application class is a parent class for the game. It is used to control the higher level events

in the game: starting a game, getting player turns, checking if the game has ended, etc.

- firstPlayerStarts : tag indicating that the first player starts the game.
- gameEnd : tag indicating if the game has ended
- changeTurn(): changes the turn to the other player
- getFirstPlayer(): get the first player : the player with the first player tag
- getSecondPlayer(): get the second player
- startGame(rematch):start a new game. Starts a rematch if rematch tag is true

Distribute visitor pattern

Distributing seeds to stores and houses was best seen to be achieved with the visitor pattern.

- lastVisitedPlace : the last visited place that the visitor has visited
- player : player side the visitor is distributing
- seeds : seeds to distribute
- visit(p) : visit a place. Place can be a store or a house.

Model-view controller for GUI

We used a model-view-controller approach for linking the modeled classes with GUI. The GUI itself is only capable of displaying the menus, the houses and stores. The controller updates the GUI after each move and connects the seed counts of houses and stores with GUI elements. Also, all relevant user actions are transferred to the Mancala model by the the controller. This way we were effectively able to separate the logic and the visual representation of the game.

Development history

By Kristjan and Lasse

October: Previous work:-by Timo, Lasse, David, Kristjan

- learning the rules of Mancala
- finding out the strengths of team members
- a simple idea about work division
- setting preliminary deadlines for subtasks
- we setup the git repository at github for the project
    - http://github.com/hyyrynen/mancala/tree/master/project/

26. October: Real work starts on the project. -by Timo, Lasse, David, Kristjan

- We meet in the class and start to work on the user story titles
- We continue the work after the class
- Results:
    - Generated 28 user story titles
    - Fixed deadlines for subtasks of the project

    - A simple class diagram
    - A plan for using design patterns -> in a separate file “design decisions”

29. October: by Kristjan
    - Corrected user story titles, highlighted the ones with the same idea / unrelated ones
    - Wrote 11 user stories, 12/31 done
    - Wrote a simple Menu / UI plan

7. November: by Kristjan
    - added file with Mancala rules
    - added 9 new user story titles

9. November: Lasse, Timo
    - some user stories
    - cleaning up the file

10. November: Kristjan
    - Finished the empty user stories
    - 29/31 should be done
    - user stories 11,12, 20 have to be added/reviewed/changed
    - set deadlines for the rest of the project

11. November: Kristjan, Lasse, Timo
    - Object diagram templates
    - First object diagrams - for user stories: 2, 5, 6, 11

14. November: David
    - Some object diagrams for user stories: 10, 13, 14, 15, 16, 17, 21, 24, 27, 28, 30, 31

20. November: Timo, Kristjan
    - Finished Class diagram design
    - Added method titles, constructors, design patterns

22. November: Timo
    - Fujaba modelling -> see github for logs

23. November: Timo
    - Fujaba modelling -> see github for logs
    - Most of Gui

23. November: Kristjan
    - Fujaba modelling -> see github for logs’
    - A little of Gui

25. November: Kristjan, Lasse, Timo
    - Fujaba modelling -> see github for logs'
    - Documentation

26. November: Kristjan, Lasse, Timo
    - Fujaba modelling -> see github for logs'
    - Documentation