Design Specification

CSE 361 - SPRING 2014

**Pachisi**

Jacob Bohac

Aaron Wollman

Joshua Grossman

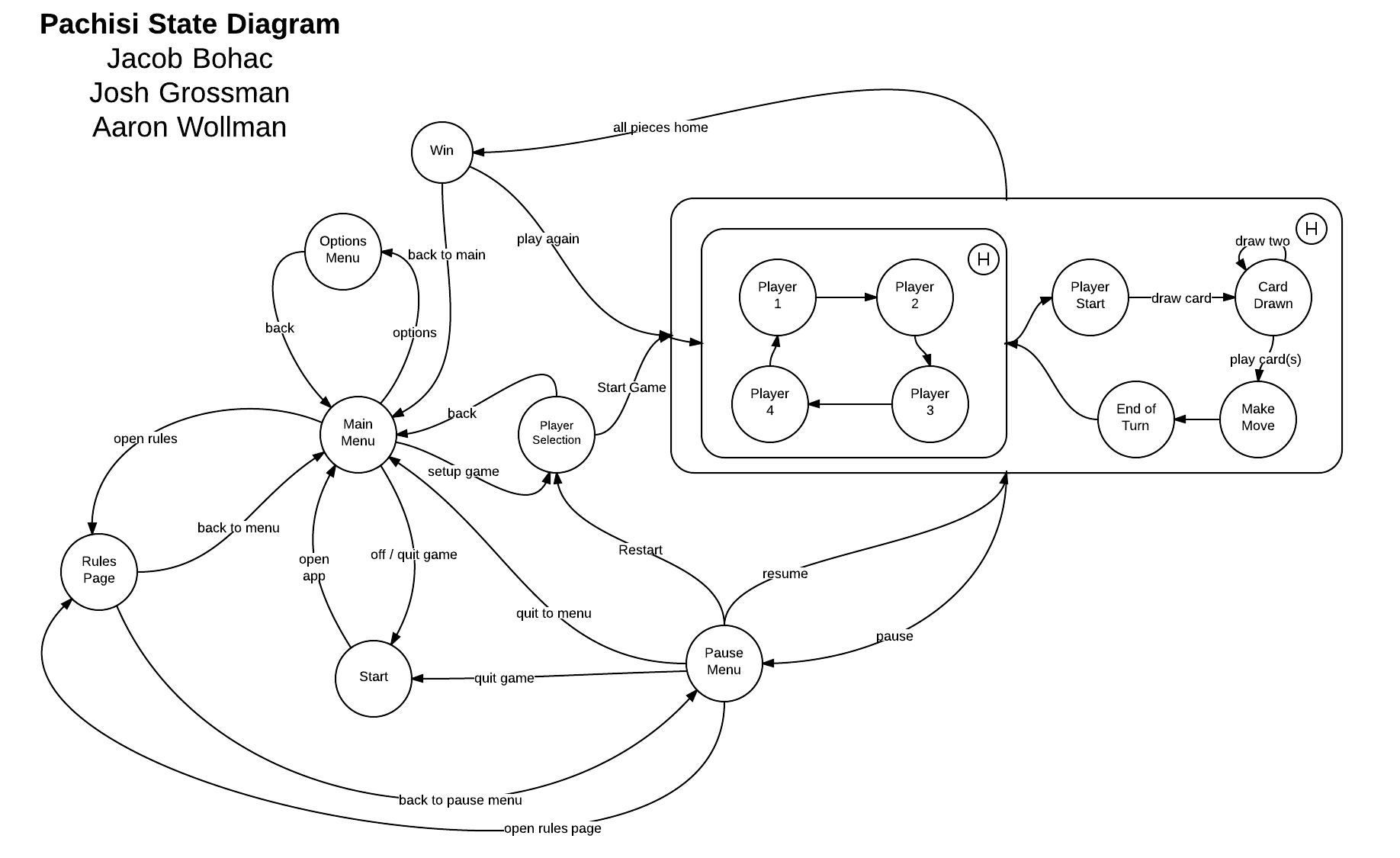
1. Introduction

This document details the design of a Pachisi system for Android devices. Pachisi is an ancient board game in which players try to get from a start place to their home base by going around a board.

One issue the system faces is in how well it translates from a physical board game into code. Another issue is that our programmers have little to no experience developing for Android devices. The programmers all know Java, however, so they may be able to adapt.

This design document contains a basic design of the system. In Section 2, the document will detail the architecture of the system. In Section 3, a Class Diagram will be described to show the system’s structure. Stage 4 will relate this document to the Requirement Document (see ‘System Requirement Specification: Pachisi’).

2. Architecture



In the ‘Pachisi State Diagram’ above, the system will begin at the ‘Start’ bubble. This state is when the user is on his/her Android Homepage. When the app is opened, the system will open in the ‘Main Menu’. From here, the user can look at the ‘Rules Page’, go to the ‘Options Menu’, or ‘Setup Game’.

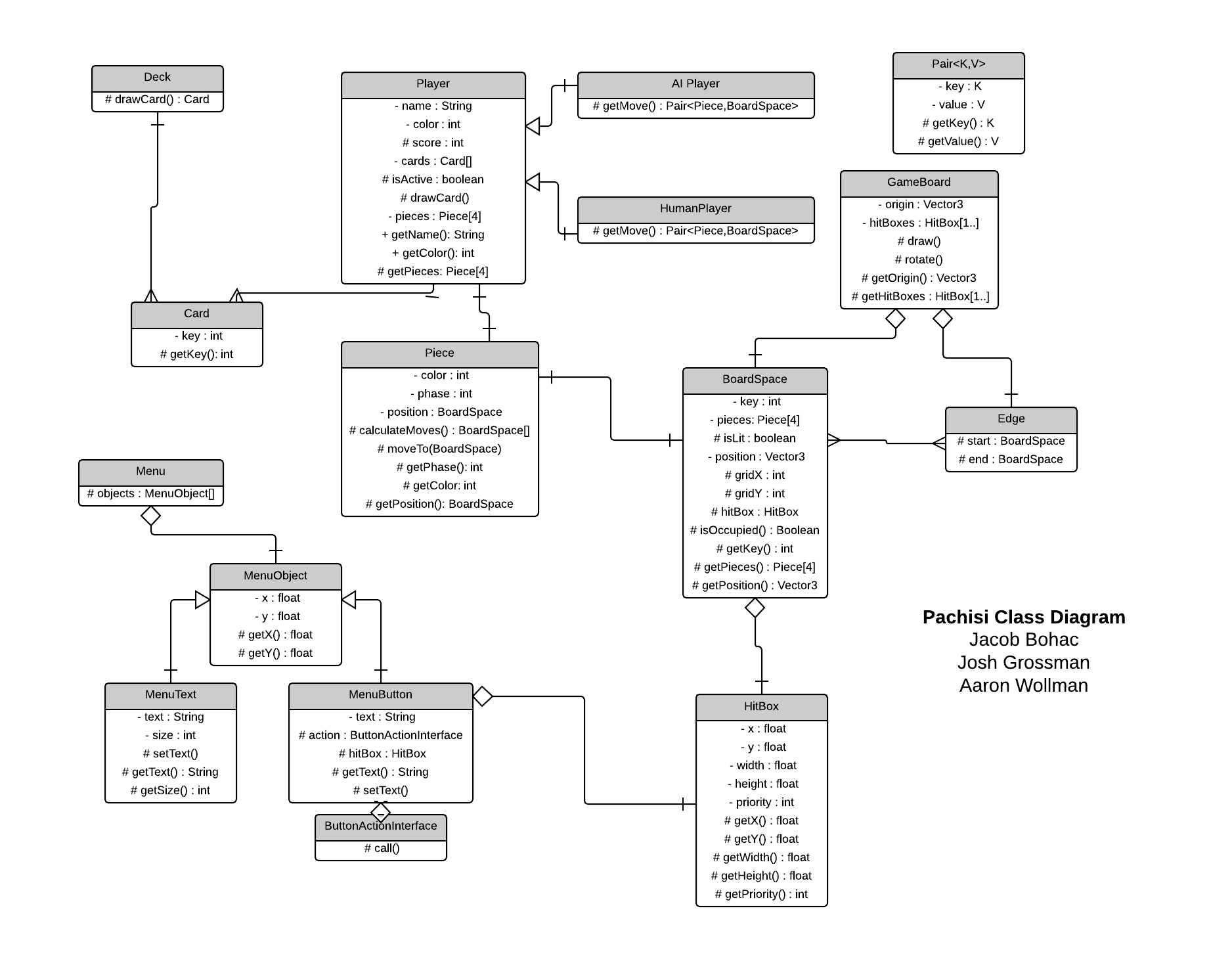
If the user goes to the ‘Rules Page’, the user can read through the rules of Pachisi. The page will then offer to go back to the ‘Main Menu’, or to the ‘Pause Menu’ depending on where the user entered the ‘Rules Page’.

Otherwise, if the user goes into the ‘Options Menu’, the user can change the system’s settings. Options are TBD at this point. The user then can go back into the ‘’Main Menu’.

Finally, the user can decide to ‘Setup Game’ and will go to the ‘Player Selection’ section. Here, the user can decide how many players will play the game, as well as designate names. Once the user is ready, he/she can select ‘Start Game’ or go back to the ‘Main Menu’.

When in the game, the players go in order (1, 2, 3, 4). When a player starts their turn in the ‘Player Start’ phase, they draw a card leading to the ‘Card Drawn’ state. If the card is a ‘draw two’, a player reenters the ‘Card Drawn’ state. One (or many) cards may be played in one turn leading to the ‘Make Move’ state, where the player is allowed to move his/her piece. Once the piece is moved, the ‘End of Turn’ phase is reached and the next player gets to make his/her turn.

3.Classes



The game board class contains an origin location that is used for drawing the board. From this origin point board spaces and game pieces game be drawn using their relative position to the game board. The BoardSpace and Edge class are a part of the underlying graph data structure that represents spaces on the game board. BoardSpaces will have identifiers gridX and gridY which represent their location in the play space, but are not associated with the relative position on the board. The edge class is used to connect sequential BoardSpaces into a sequential path that can be traversed. Each BoardSpace will also contain a HitBox. This will be used to identify which box is being selected on user input. The collection of these HitBoxes is contained in the GameBoard class and will contain all possible BoardSpaces that the user may select. BoardSpace will also contain the set of all pieces located at that space, that way the user can select their piece to move using the hitBox associated with that BoardSpace.

The Menu class is composed of an array of MenuObjects. Each MenuObject will be positioned on the screen in relation to its position in the ‘objects’ array in the Menu class. A MenuObject is an abstract class that has two subclasses: MenuButton and MenuText. MenuText is a simple container class for adding text to menus while the MenuButton class allows for user interaction. When a user presses a MenuButton, the associated ‘action’ will be called. An ‘action’ is a class which implements the ButtonActionInterface. A class implementing the ButtonActionInterface must have a ‘call()’ function which performs the action that the specified button should accomplish. To determine which button is pressed, the collection of HitBoxes in the GameBoard class is used.

4. Requirements

The design will meet our requirements through a variety of ways. At this phase, the document doesn’t guarantee the performance and hardware requirements stated in 3.2, 3.4.3, 3.4.4, and 3.4.5. This document does guarantee, however, that the program will be written in an Object Oriented style as stated in 3.3.1.

Section 2 of the document describes how the system transitions between states to allow the user to play a game of Pachisi, as stated in 3.4.1. Using the classes and functions of Section 3, the system will follow the rules of Pachisi, as stated in Section 4.1.2. As shown in the State Diagram, the Appendix rules will be accessible from the ‘Main’ and ‘Pause’ menus. This also meets Requirements 3.4.6 and 4.1.3 by including a ruleset for the player to read.

In Section 2, the ‘Pause Menu’ can be called anytime when the game is started. The ‘Pause Menu’ then allows the user to quit, restart, and continue, which are shown in the diagram as state transitions. This satisfies Requirement 4.1.5.

As shown in Section 3, the design allows both human and AI players by having the player types be subclasses of the ‘Player’ class, as required by 3.4.2, 4.2.1, and 4.1.1 in the Requirements. The ‘Human’ class will be focused on player input. On the other hand, the ‘AI’ class will contain the logic the system’s computer players will use to move around the board. This ‘AI’ can be improved by changing the ‘getMove’ function. By adding an integer to be passed to the AI’s ‘getMove’ function, we can adjust the ‘AI’ difficulty to different levels. This satisfies requirement 4.3.1.

For 4.3.2, we can implement JPA and SQL to apply an online multiplayer to the system. We could also use the Android libraries to implement bluetooth multiplayer, such as described in 4.3.7.

In the options, we can have allow drop in/out multiplayer (4.3.8) by having an option in the ‘Pause Menu’. When chosen, the human player will be changed to an ‘AI’ which will carry all of the human’s variables for the game.

GUI improvements (4.3.3) and sound (4.3.4.) can be implemented by adjusting the ‘MenuObject’, ‘GameBoard’, and ‘BoardSpace’ classes. Music can be placed in these classes to run when in the the menus and during the game. The ‘BoardSpace’ class can be improved to provide feedback to the player (4.3.9).

We can adjust the Gameboard class to allow the game board to zoom and rotate as stated in 4.3.11. We can add an ‘ActionListener’ to allow the user to touch to implement the zoom and rotate.

For 4.3.5, the system can create an XML, JSON, or a .txt file to save data when the user hits ‘Quit’ or exits out of the app. This file will hold the required save data for the game. When the system is reopened, an option is presented to the player to implement the save file.

Requirement 4.3.10 can be easily implemented by enabling the gyroscope using the Android SDK. Part of the card class will be removed in favor of a random number generator. A dice roll animation will also be implemented in the system.