Software Design Document

for

Force Connector

Version 1.0 approved

Bethany Roberts, Brandon Shelton, Payton Long, Cole Dalfonso, & Casey Sanchez

Skywalker LLC

2.10.2021

Table of Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed

1. Introduction

1.1 Purpose

Within this document are the System Design Description for the Force Connector game/software. Each component and architectural design that will be used in development of this game/software and how each system and component will interact with each other.

1.2 System Overview

Force Connect shall be developed and written in Java and the development team will follow the MVC pattern. Using the MVC pattern will allow the development team a better insight into how the game/software and its components communicate and interact.

1.3 Definitions, Acronyms and Abbreviations

"Player" & "User"	Human player/user of the game/software
Development Team	Direct team creating the software/game
PvP	Player VS. Player
GUI	Graphical User Interface
System	The software/game as a whole
AI	Computer controlled "Player"

1.4 Supporting Materials

1. Force Connect follows the general rules of Connect 4 Below

    a. https://www.officialgamerules.org/connect4

2. Parts of this SDD document was created using StarUML

    a. https://staruml.io/download

1.5 Document Overview

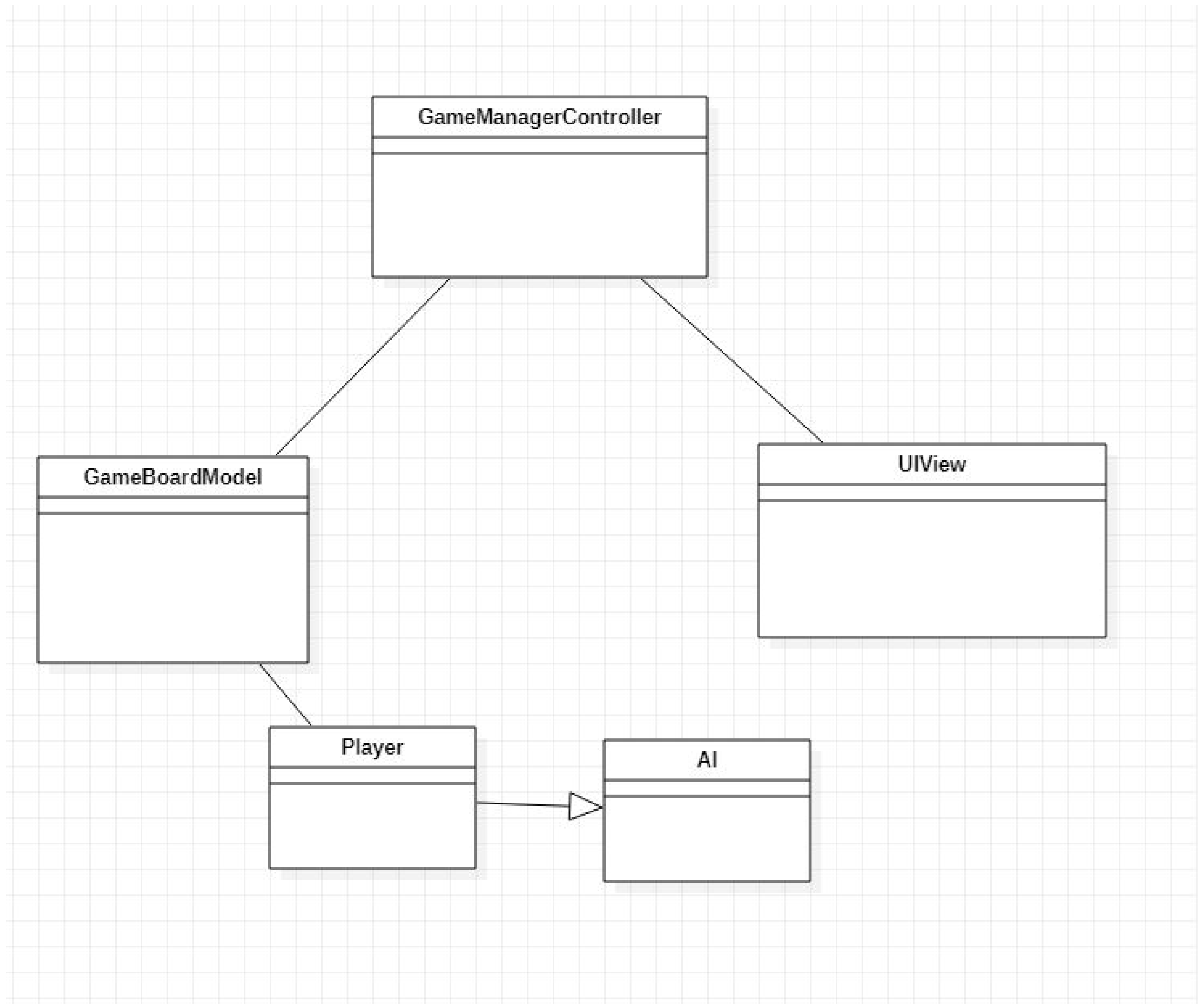
Section 1 will outline the document as a whole and specify definitions used.

Section 2 will provide the class architecture of the program.

Section 3 will provide a State Diagram and a Sequence Diagram.

2. Architecture

The following diagram describes the overall design of the system including an overview of the relationship between the key components and classes.



2.1 Overview

This system utilizes the common software architectural pattern, Model-View-Controller ("MVC Architecture"). The unpredictable rate of exchange to and from a user justified using the pattern. It is described in further detail in Figure 1.

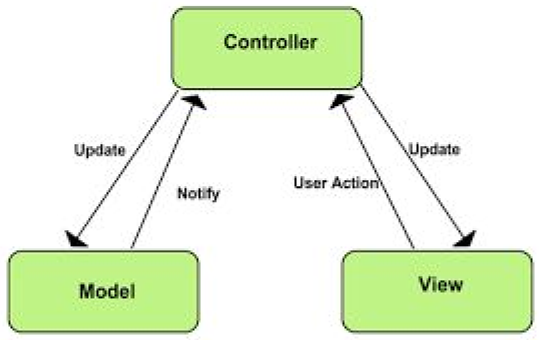


Figure 1: Model-View-Controller Diagram

2.2 Component 1..5

2.2.1 Game Manager Class

The Game Manager (GM) class acts as the controller component which handles the request flow in the Model-View-Controller diagram shown in figure 1. This component does not handle any logic, but sends requests to the model to handle the logic. The controller does however handle what to do with the results of the logic. The game manager will also get information from the view.

Game Manager Class
+aiTurn: bool +playerTurn: bool +columnFull: bool +isGameOver: bool +firstPlayer: char +winner: char +winStreakVal: int +currentPlayersTurn: char +column: int
<<constructor>> +GameManager(p1:int, p2:int) +startGame(): void +startApp(): void +exitApp(): void +aiTurn(): bool +playerTurn(): bool

+columnFull(): bool
+isGameOver(): bool
+randomlySelectPlayerOne(): char
+endGameWinner(): char
+endGameTie(): bool
+setWinStreakValue(): int
+getWinStreakValue(): int
+endOfTurn(): bool
+switchTurn(): bool
+cellSelection(): int column

2.2.2 Game Board Class

The Game Board (GB) class acts as the model component which handles the logic of the MVC design. The model component will never interact with the view component, but only with the controller.

Game Board Class
-gui: JPanel
-columns: JButton
-board: JPanel
-message: JLabel
-redPiece: Image
-blackPiece: Image
-noPiece: Image
~listener: ActionListener
<<constructor>> ~GameBoard()
+initializeGui(): void
+getBoard(): JComponent
+getGui(): JComponenet
+initializeActionListener(): void
+main(): void
+startGameInterface()
+startEmpty7By6Board()

2.2.3 AI Class

The AI Class contains the operations for a CPU user. There can be only 1 of these objects in the system.

What moves the AI can take.

AI Class
-myColor: char

-board: char[*,*] -turnComplete: boolean -dest: int[*] -selected: int[*]
<<constructor>> +AI() +aiRandomMove() +getColor(): char +getTurnComplete(): boolean +setTurnComplete(): (c: boolean) void -move(): void +getMove(boardState: char[*,*], row: int, column: int): char[*,*]

2.2.4 Player Class

The Player class contains the operations for a human user. There can be only 1 of these objects in the system.

What moves the Player can take.

Player Class
-myColor: char -board: char[*,*] -turnComplete: boolean -dest: int[*] -selected: int[*]
<<constructor>> +Player(color:char) +getColor(): char +getTurnComplete(): boolean +setTurnComplete(): (c: boolean) void -move(): void +getMove(boardState: char[*,*], row: int, column: int): char[*,*]

2.2.5 User Interface Class

When a user opens the Force Connector application, a GUI shall be displayed. This GUI shall use the observer and observable design patterns in order to incorporate mouse clicks as a way of registering user input. A representation of what the display shall look like is shown below. This class gets information from the controller and displays the information to the user. This template will dynamically render HTML based on the data feed into it.

Anything that the user directly interacts with, it should only be the UI, is the view. The UI doesn't change any data itself but instead sends that request to the controller.

User Interface Class
-coordinateX: int -coordinateY: int
<<constructor>> +UI() +playGameButton(): JButton +mainMenuButton():JButton +rulesButton(): JButton +exitGameButton(): JButton +selectColumnButton(): JButton +playAgainButton(): JButton +backToMainMenu(): JButton +selectDifficulty(): JButton +exitToMainMenu(): JButton +displayRulesText(): JButton +exitRulesToMainMenu(): JButton +fireworksAnimation():void +sadFaceAnimation():void +winningSound():void +losingSound():void +trophyDisplay():void +winLossTieDialogue():void +addPieceToBoard():int X, int Y

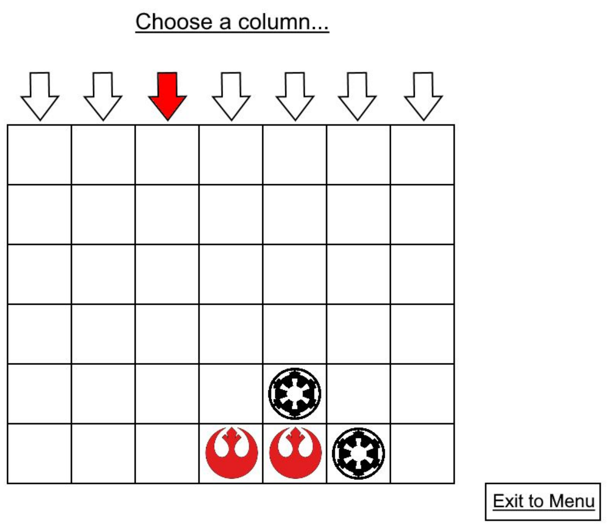


Figure 1: GUI when playing Force Connector.

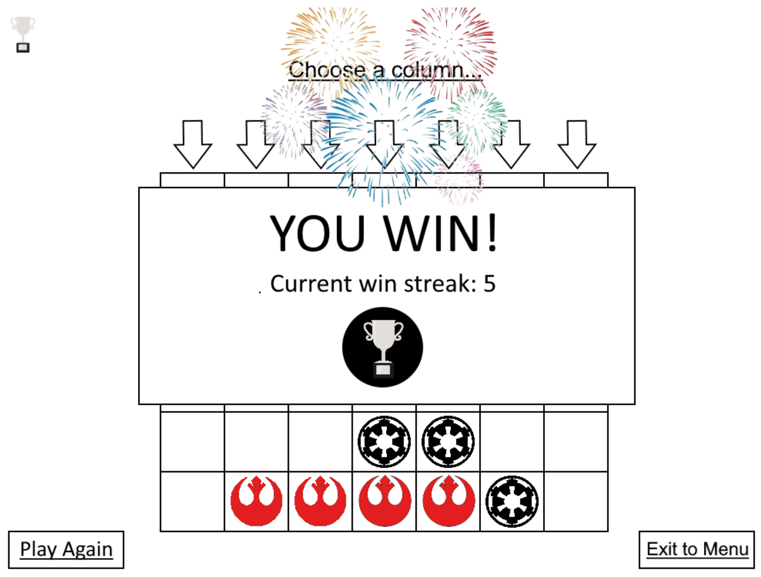
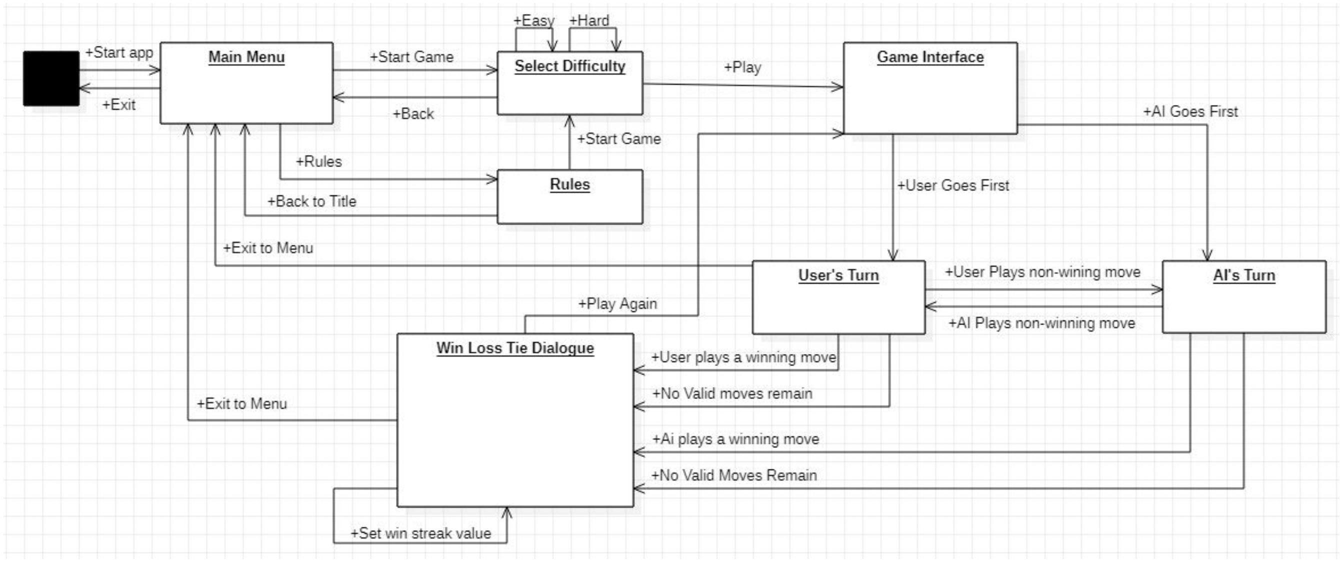


Figure 2: GUI when a game of Force Connector is won.

3. High-Level Design

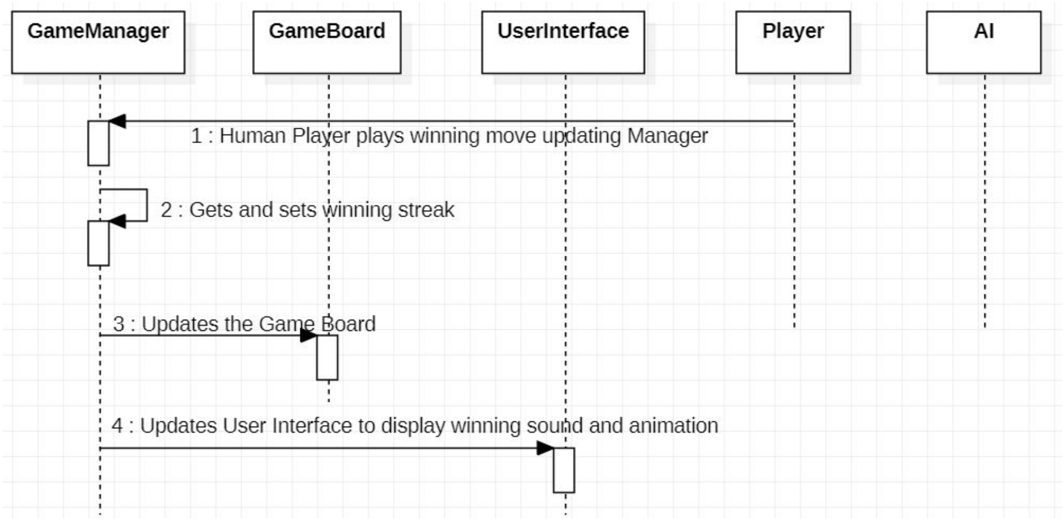
2.3 State Diagram

This state diagram includes all possible actions that the user can take to interact with the system and the flow of those actions. Some actions are not done by the user (eg: randomly deciding the first player) but the result of those actions influences the user's possible interactions.



3.2 Sequence Diagram

The Sequence Diagram below displays the interaction between classes when the Human Player plays a winning move.


GameManager
GameBoard
UserInterface
Player
AI
1 : Human Player plays winning move updating Manager
2 : Gets and sets winning streak
3 : Updates the Game Board
4 : Updates User Interface to display winning sound and animation