Deliverable # 3

Group # 13

Daniel, James, Dylan and Zsanett

*Project Topic*

For this deliverable our goal was to have a functional game that is playable through the console, as well as packaging our code for better maintainability and readability. We have refactored much of our code that we had written from the previous deliverable for maintainability.

Since last deliverable we have added a few new implementations; An AIPlayer class, a HumanPlayer, a Game and GameLauncher class. The purpose of the AI player is to allow the user to play a single player game. Currently the AI can only make random moves, making it an easy AI to play against. The Human class is not an AI, rather it would be an opponent that you’re playing with. This class is a child class of the Player class, from deliverable two. HumanPlayer will be assigned a color, and will do the appropriate checks for moves. The Game class will create a game and players, the players will be chosen depending on the users’ choice upon launching the game. Also, the class will determine when a turn starts and ends or when a game starts and ends. Finally, the GameLauncher class simply launches a game.

The game, as stated previously, is played through the terminal. Upon launching the game, you will be prompted if you wish to play single player or multiplayer. Once you choose, you are assigned as red; at the moment it is represented as “R” on the terminal. The user will be asked to pick the location of a red piece and move to a valid location. For location coordinates we use a (x,y) coordinate system.

All the other classes have been refactored to fit our new structure.

Class #1 – AIPlayer

By James

Design: The AIPlayer, child of Player, is a class that contains relevant data for an NPC. The AIPlayer would be considered an easy opponent because it only has the ability to make random moves.

|  |
| --- |
| Game |
| - generator: Random |
| + RandomStart(): Location  + RandomEnd(): Location |

Class #2 – HumanPlayer

By James

Design: The HumanPlayer, child of Player, is a class that contains relevant data for a human opponent. The HumanPlayer has the appropriate data relevant to an opponent, such as color and move validations.

|  |
| --- |
| HumanPlayer |
| - scanner: Scanner |
| + takeInput(Boolean pieceSelection): Location |

Class #3 – Game

By James

Design: The Game class, which takes on the singleton pattern, will initiate a game. It contains methods that will set the players depending on the input of the user. Also it will initialize a board and determine turns, based on whether the user is a human or an AI.

|  |
| --- |
| Game |
| - instance: Game  + blackPlayer : HumanPlayer  + redPlayer : HumanPlayer  - board : Board  - mode : int  - gameOver : Boolean |
| + getInstance(): Game  + getMode() : Integer  + initialize() : void  + turn(HumanPlayer aPlayer) : void  + turn(AIPlayer aPlayer) : void  - canMove(Player aPlayer) : Boolean  - gameOver(Player aPlayer) : void  + gameOver() : boolean |

Class #4 – GameLauncher

By James

Design: The GameLauncher class will retrieve the single instance of Game and initialize it. All this class is responsible for is initializing a game and managing the turns.

|  |
| --- |
| GameLauncher |
|  |
| No Methods or variables. |