**HERO BATTLES**

Computer Science 1101 Project

Craig Aucoin, Sam Irvine, Blake Patriquin, Mariah Pinto & Lindsey Power

Overview of game concept and elements

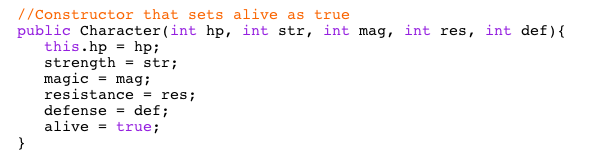
We have designed a turn based role play game that allows users to control the actions of their characters as they encounter obstacles in the quest to reach the castle guarded by their opponent. Two players are able to play the game at one time, passing the controls between each turn. Players will select 3 characters each, based on their skill sets and will begin with their characters in front of their castle (in opposite corners of an 8 by 8 game board). Players can move twice per turn, strategizing to guard their castle and also advance to take the opponent’s castle. Encountering a character controlled by the opponent will result in a duel; attacks are dictated by the attributes of the character involved. These encounters will result in a loss of health points for one or both of the characters. The object of the game is to move at least one character all the way across the game board and take the opposing player’s castle without being defeated by attacks from the opposing characters. A victory results in more advantageous attributes for the character which will be updated and stored at the end of the game.

A game board is composed of a series of 2 dimensional array LinkedLists made up of unique Nodes that can hold either a Character, an Obstacle (such as water, a rock or a hole that the character must navigate around), an Objective (the goal points at opposite ends of the board) or an Empty Space. Each space on the board is coded by a position in the array and a LinkedList (with 8 LinkedLists in total, one for each row of the game board). This structure was used in order to allow the implementation of different object types into the board. The generation of the game board begins by filling the board with empty spaces and then randomly generating obstacles throughout the board. In any given space on the board, the chance of an obstacle being generated is 10%. Objectives for each team are then generated in opposite corners of the board and three characters for each team are generated in the positions surrounding their team’s objective. Characters are able to move one space in any direction, but are not able to move into spaces that contain an obstacle, and must navigate around the obstacle.

GUIs are used to create an attractive interface where the positions of the character’s relative to one another and to the obstacles are clearly visible. The interface allows the player to click on a character and click on where they would like the character to move to move the character about the game board. GUIs are also used to present the selected character’s stats to the left of the game board, and to generate a start menu at the beginning of the game. When the characters are selected at the beginning of the game, the stats are uploaded from a file, and at the end of the game the updated stats of that character are then saved to the file once again to be accessed in future games. This allows the player to save characters over a series of games.

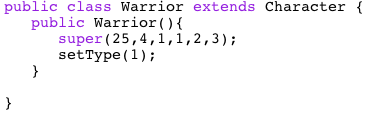
Summary of classes

A Character class has attributes for the type, name and image of the character as well as all of the stats for the character (including level, hp, strength, magic, resistance, defense and XP) which determine the abilities of the character to fight other characters, and a boolean variable that signifies whether or not the character is alive. There are get and set methods for all of the attributes of the character. The Character class also has defend and attack methods that take into account the type of character and the character that they are opposing.



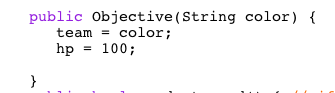
*Constructor for the Character class*

Mage, Tank, Warrior and Ranger classes inherit from the the Character class. Each one creates an instance of the super class with different values for the stats attributes of the Character. The constructor also sets the character type.



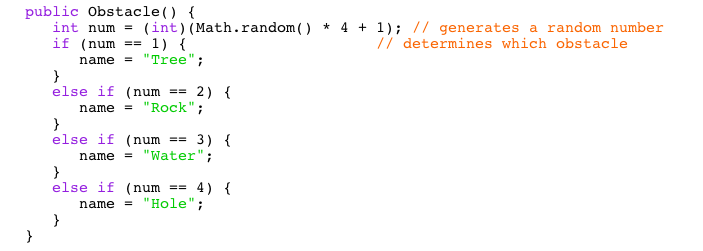
*A sample of one of the Character type classes*

An Objective class has attributes for colour and hp, with a constructor that takes in a value for colour (based on the colour corresponding to the team) and sets the hp to a default value. Includes methods that check if the objective has been attacked or destroyed.



*Constructor for the Objective class*

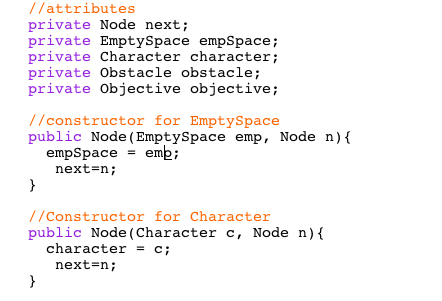
An Obstacle class has a name attribute. A number between 1-4 is randomly generated by the constructor and corresponds to one of four possible obstacles whose name is then set in the constructor.

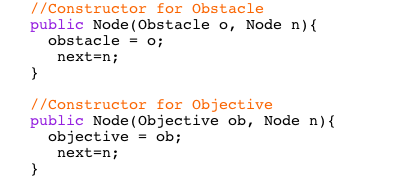


*Constructor for the Obstacle class*

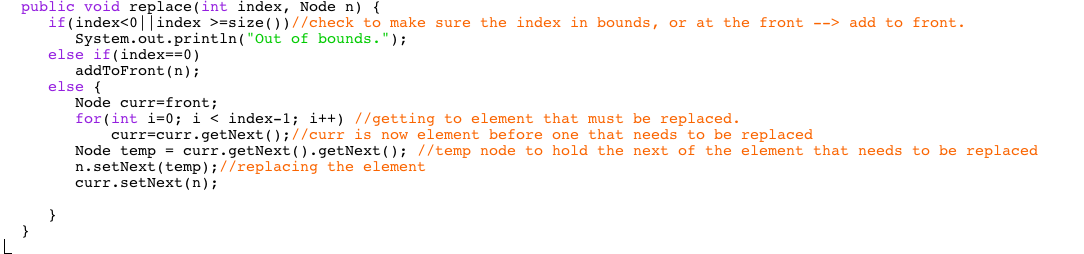
An EmptySpace class has a data attribute that is set to a default value in the constructor.

A unique Node class allows the creation of Nodes that take in a Character, an Objective, an Obstacle or an EmptySpace such that each Node on the board can be filled with a different object type. A LinkedList class includes the methods learned in class as well as a “replace” method that replaces a Node at a given index with a specified Node.



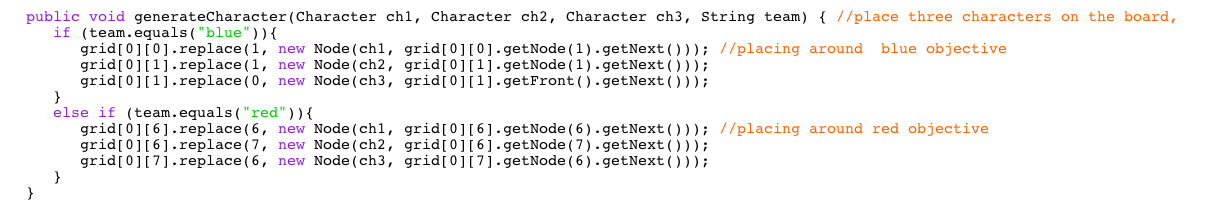


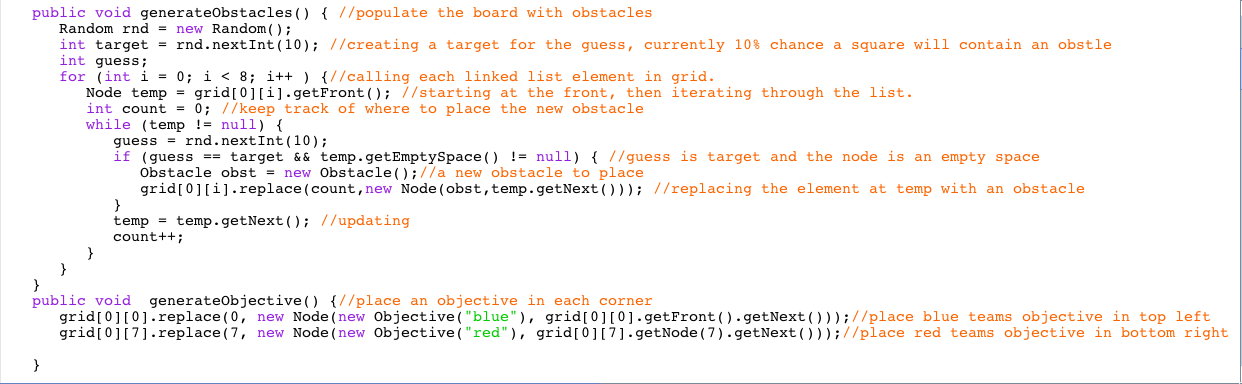
*Shows the attributes and constructors that are different for each of the object types that can be taken into a Node*



*The above shows a unique method that replaces a Node at a given index with a specified Node*

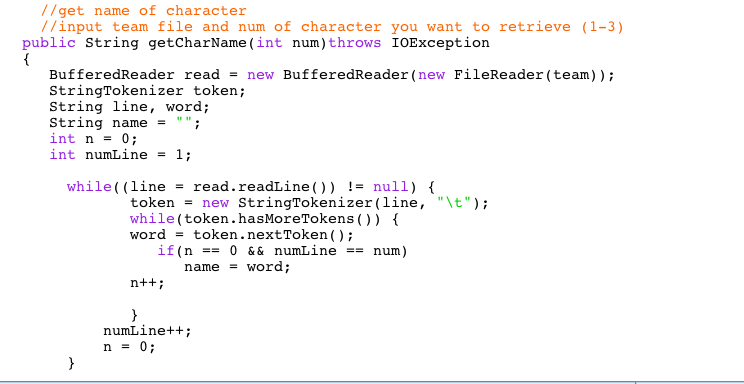
A GridMap class has a double array LinkedList attribute called grid and aggregates the Objective, Character, Obstacle and EmptySpace classes. The constructor fills the grid (each LinkedList) with EmptySpace objects. The class has methods to generate Characters and Objectives in fixed locations as well as a method to generate Obstacles in random locations. A moveChar method is also included that locates the selected character and moves it in the specified direction using if statements for each direction that the character is able to move.





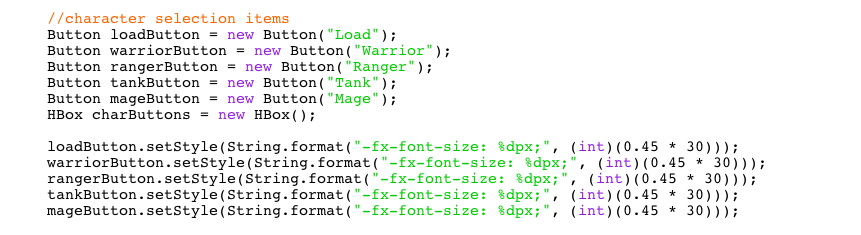
*The code above shows the methods to generate all of the elements of the grid map*

A GameFile class saves character’s names and type to a file. On the start menu of the game there is an option to load previously created character which will call the method to access characters from a text file and load them into the game. Characters selected for each team will be saved into a text file to be accessed in future games.

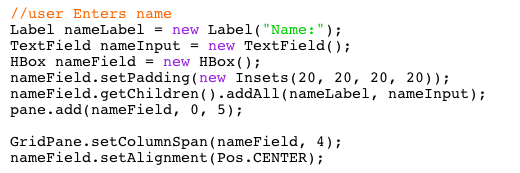


*Reads in the name of a character from a file*

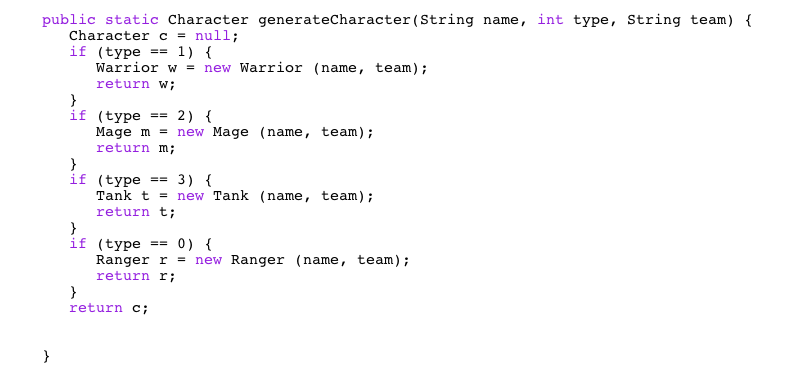
Finally, the FullGame class aggregates the aforementioned classes and incorporates GUIs to create an interactive game for the user with a variety of elements. This class uses GUIs to generate a start page that allows the user to select which characters they would like to use on their team, name them, and load the character into the game. The class then uses imported images and javaFX functions along with the methods from the GridMap class to generate a visual representation of the game board. A grid type game board with representative images in the locations of characters, obstacles and objectives is generated, with a vertical box that displays messages, such as the character selected and the team turn, and dynamically changes colour as the team’s turn switches. Each space that contains a character is a usable button, and when pressed, the spaces surrounding the selected character that are valid movement positions become usable buttons and are highlighted to allow the user to easily determine where they are able to move. Pressing one of the highlighted spaces moves the character to that location. There are also methods to process attacks on characters as well as attacks on the objectives and reduce the hp of the object being attacked based on the attributes of the character attacking. A character is killed when it’s hp reaches 0. The game exits when the hp of one of the objectives gets to 0, or all of the opposing team’s characters are killed.



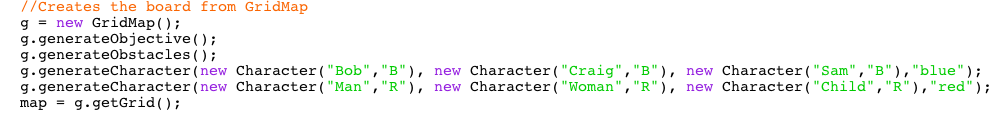
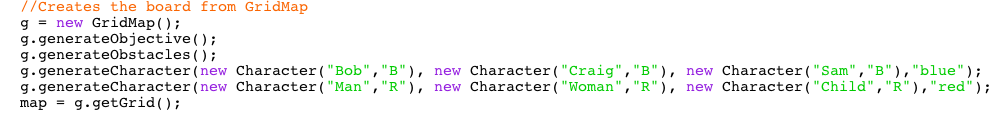
*creates buttons that the user can press to select the characters they would like to use*

**

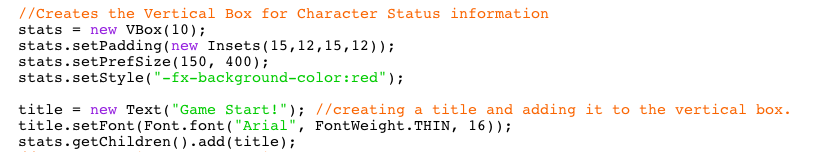
*Allows user to name the character by entering a name in the text field*

**

*Generates a character to be used in the game*

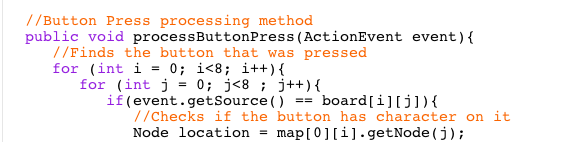


*Uses the methods from the GridMap class to generate a board*

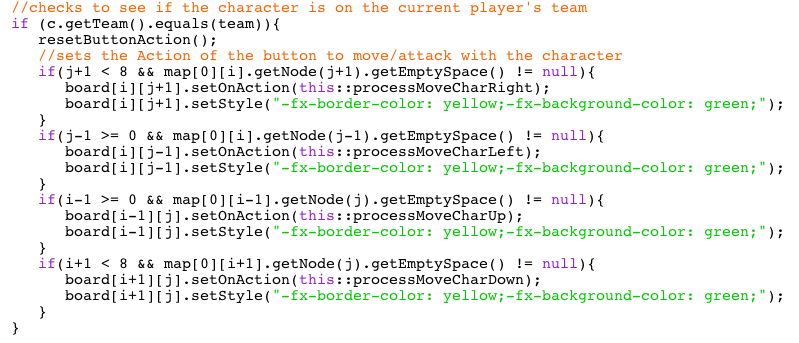


../../../Desktop/Screen%20Shot%202017-04-14%20at%2012.55.04%20PM.png

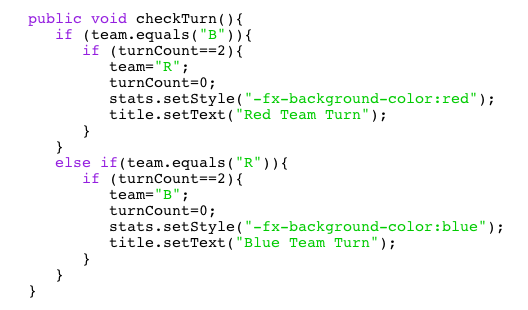
*Creates the vertical box that is initialized to red (to represent red team’s turn) and displays messages*

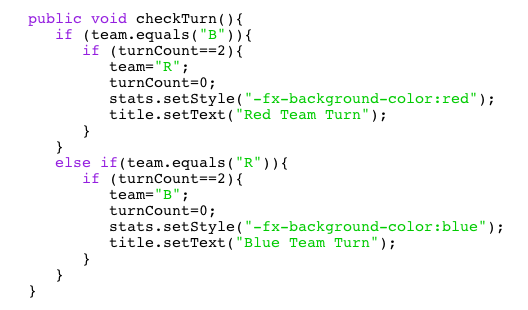


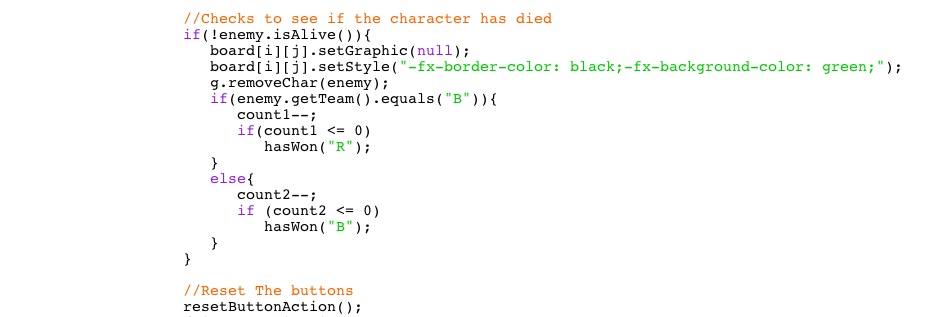
*Finds the button that was pressed and checks if it’s a character*

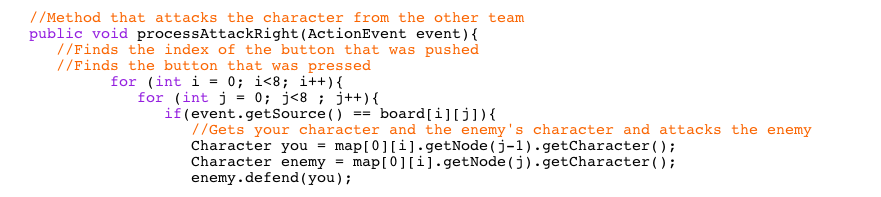


*If the selected character is on the team whose turn it is, the spaces surrounding the character become usable buttons*

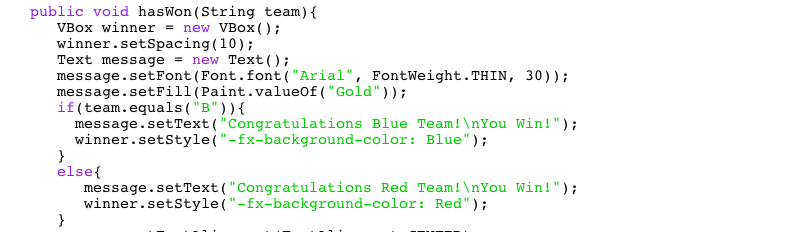
**

**

*The checkTurn method is called after each move and switches turn to opposing team after two moves*

**

*Attacks a character to the right. Similar methods for attacking characters in each direction*



*A method to display a winner screen if a player has won*

Division of Work

Character/ character inheritance classes: Craig

EmptySpace/ Obstacle/ Objective classes: Blake

Node/ LinkedList classes: Lindsey

GridMap class: All members (large part by Sam)

GameFile class: Mariah

StartMenu (within FullGame): Sam/ Blake

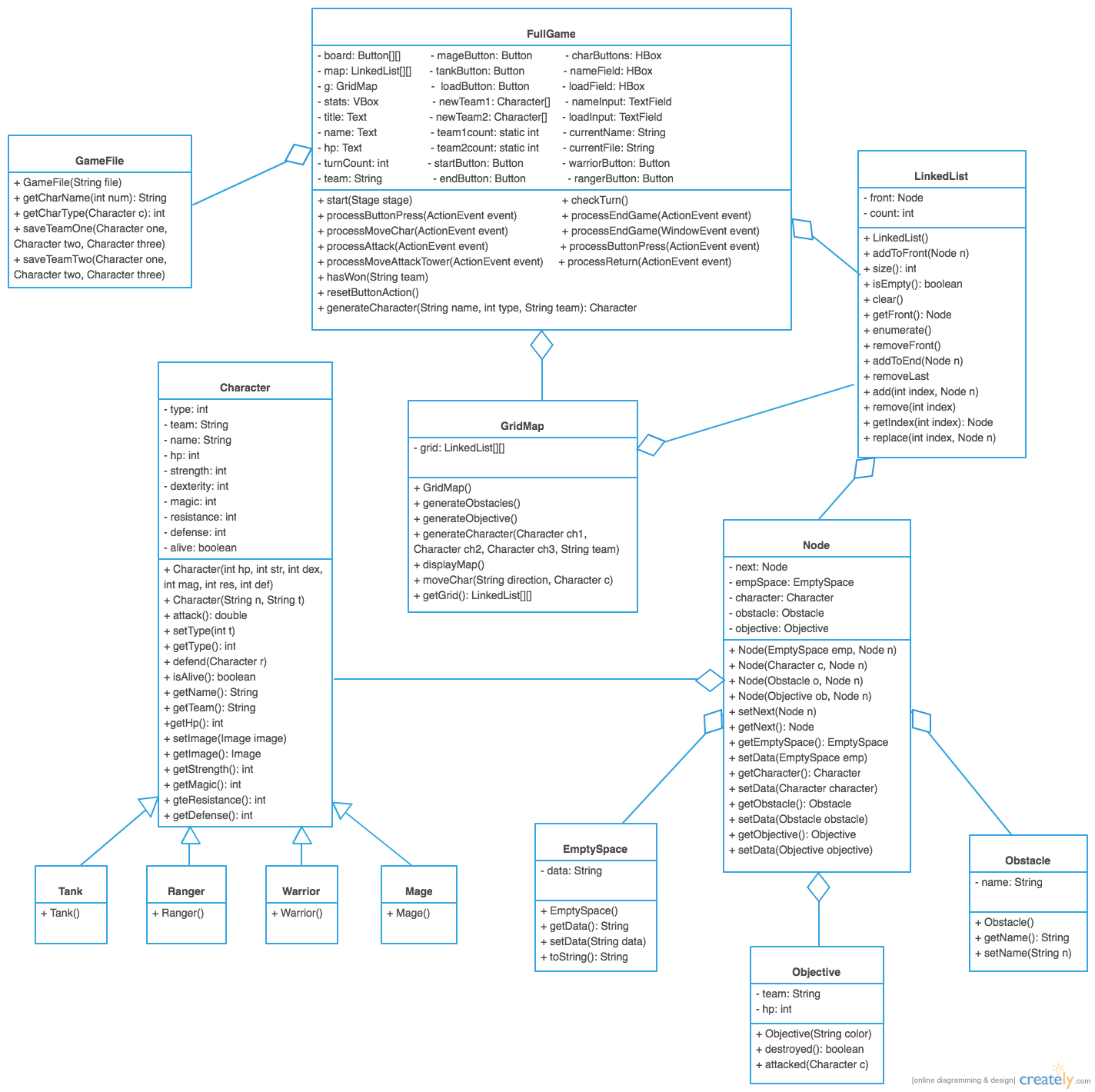
FullGame class: All members (large part by Craig)

Pixel Art: Mariah

Written Report: Lindsey

Much of the work on the larger classes was done at group meetings, allowing the contribution of all members.

UML Diagram

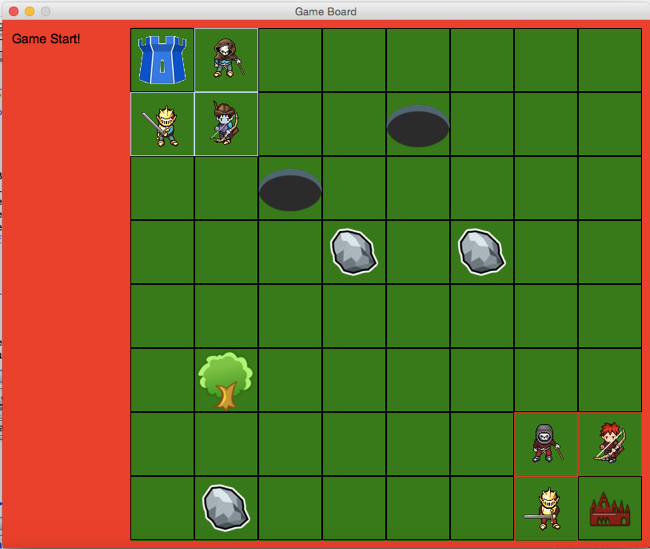


Appendix: Sample output



*A start menu displays buttons with options to start or quit If “Start Game” is selected this menu will appear, and each player*

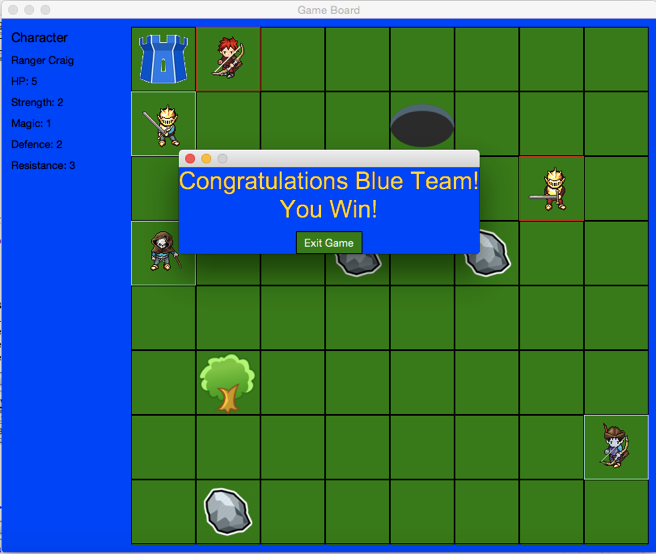
*The game will choose three of the four character types, or load previous characters*

**

*This screen is presented when the game is launched When a character is selected, stats are presented on the left and legal spaces to move to are highlighted*

**

*Background colour changes color when turn switches When a character is attacked until its hp reaches 0, the character is removed from the game board*

**

*Displays a message to congratulate the winning team if the Displays winning message if all opposing characters are killed*

*team destroys the castle*