Name: Maxine Hartnett

Project Description: Turtle graphics is often used to teach beginners the basic concepts of programming. I have turned this into a game, where one or more players submit a series of moves for different colored turtles. These moves include turning x degrees and stepping forward x units. Each turtle leaves a trail over the play field, and if another turtle runs into that trail, that user loses. The players enter their moves with a simple programming language, and the program loops through the directions until a player has lost the game by hitting another player's trail. This game could introduce people to basic programming in a fun and easy way.

Completed Requirements:

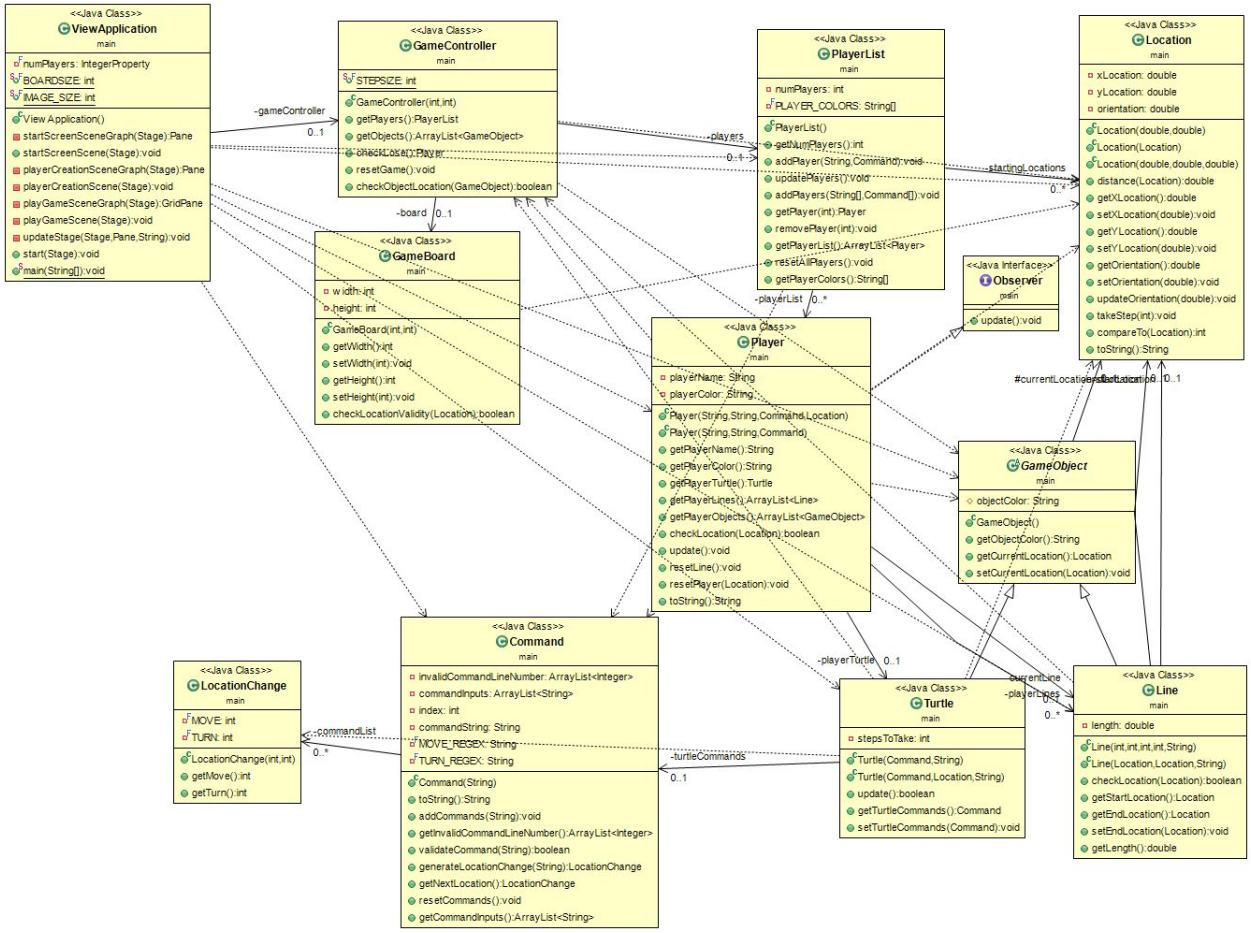
Requirement ID	Requirement
1a	Will accept commands from players
1b	Validate command structure
2a	Display game board
2b	Assign colors to players
2c	Create turtles for each player
3a	Execute commands to move turtle
3b	Update display

Unfinished Requirements:

Requirement ID	Requirement
4a	Check for win on each turn
4b	Notify player of win

Rather than notify players of a win, my program will notify them of a loss.

Final Class Diagram:

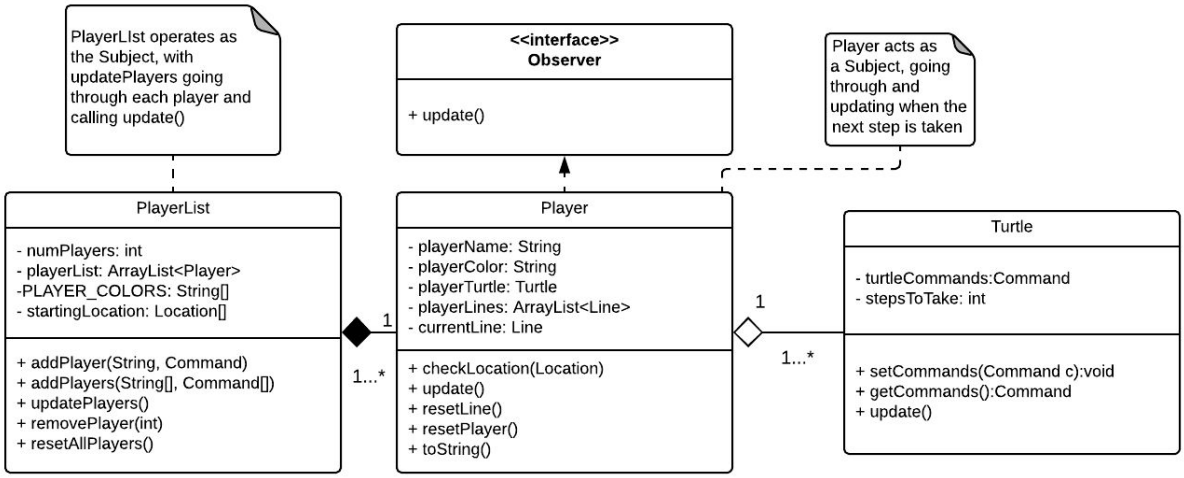


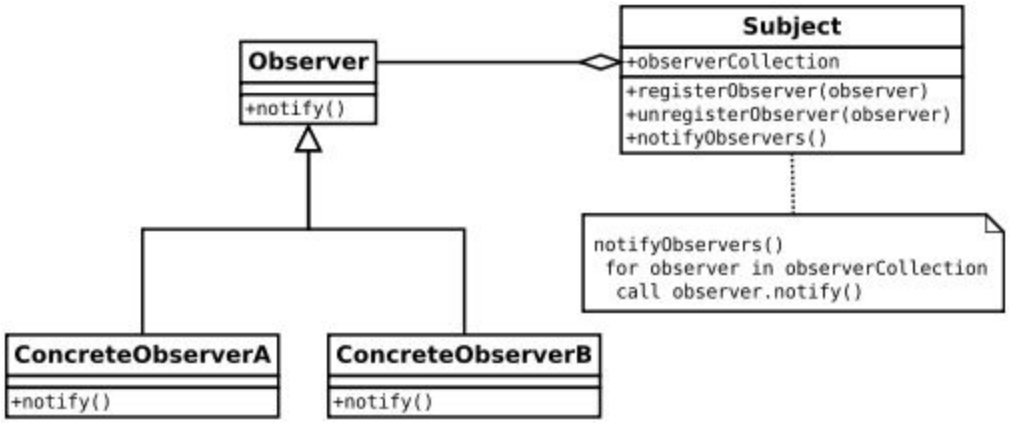
I added a few classes when I noticed that there was a lot of code building up in a class that wasn't directly related to the goals of the class. Particularly, in GameController, I saw that I had a lot of code dedicated to the creation and maintenance of a list of Players that was cluttering up the class. I moved this code into a PlayerList class, which then acted as the Subject for my Observer design pattern. I also removed a few classes I didn't think were necessary, like the PlayerInfo class, because the functionality and information contained in that class wasn't enough to justify pulling it into a class separate from Player.

However, the overall structure of my UML diagram stayed fairly consistent, although the GameController ended up being the class interacting with the ViewController UI class. Location also ended up being used by a lot more

classes, although there were still only a few strong connections between the GameObjects and the Player class. I was surprised that my original design stayed so close to my end design, but I did work pretty closely with the design when I was first setting up my classes and making the basic functionality. It really helped me to have a big-picture system when I was making the system, because I could stay on track without getting distracted or confused about my overall design.

Design Pattern





I implemented a version of the Observer design pattern. I ended up making a few `update();` methods when I was writing the code to update the location and lines of each Player's turtle. This led naturally to the Observer design pattern. I wanted to decouple the steps of the players from the nitty gritty of the turtle and line movement. The PlayerList, which acted as the

Subject, didn’t need to know what each Player object is doing during each step, only that they are doing it. I already had the methods to add and remove players for other requirements, so I didn’t implement that exactly the way the pattern does, but besides switching the `notify()` method to an `update()` method, it’s close to the official version of the Observer pattern. This pattern suits my methodology, because now if I want to add additional items that update for each step of the program, like for example a counter of the number of turns taken, it will be simple to add them to the existing methods.

What I Learned

I learned a lot from this project, as I had never done such formal design for a project up front. I found that it was useful to create a system that I could look back to for figuring out my exact plans for the project. When I had to make minor changes, it was simple to update, and having the outline of what each class’s responsibility was really helped when I was starting.

I also learned that it was good to have some flexibility in the system. Since I didn’t know much about Javafx when I started, I left that area of the UML diagram very vague, and it helped me later when I could do all my UI work in one area and only connect to the rest of the backend through one class, instead of having to juggle a lot of half-finished classes at once. However, if I were to do this project again, I would have focused more on my design patterns during the initial design phase, as it was hard to go back and work them in, and I think I would have benefited from having that knowledge up-front.