Niketh Gorla, Natalie Tobiason, Ethan Schultz

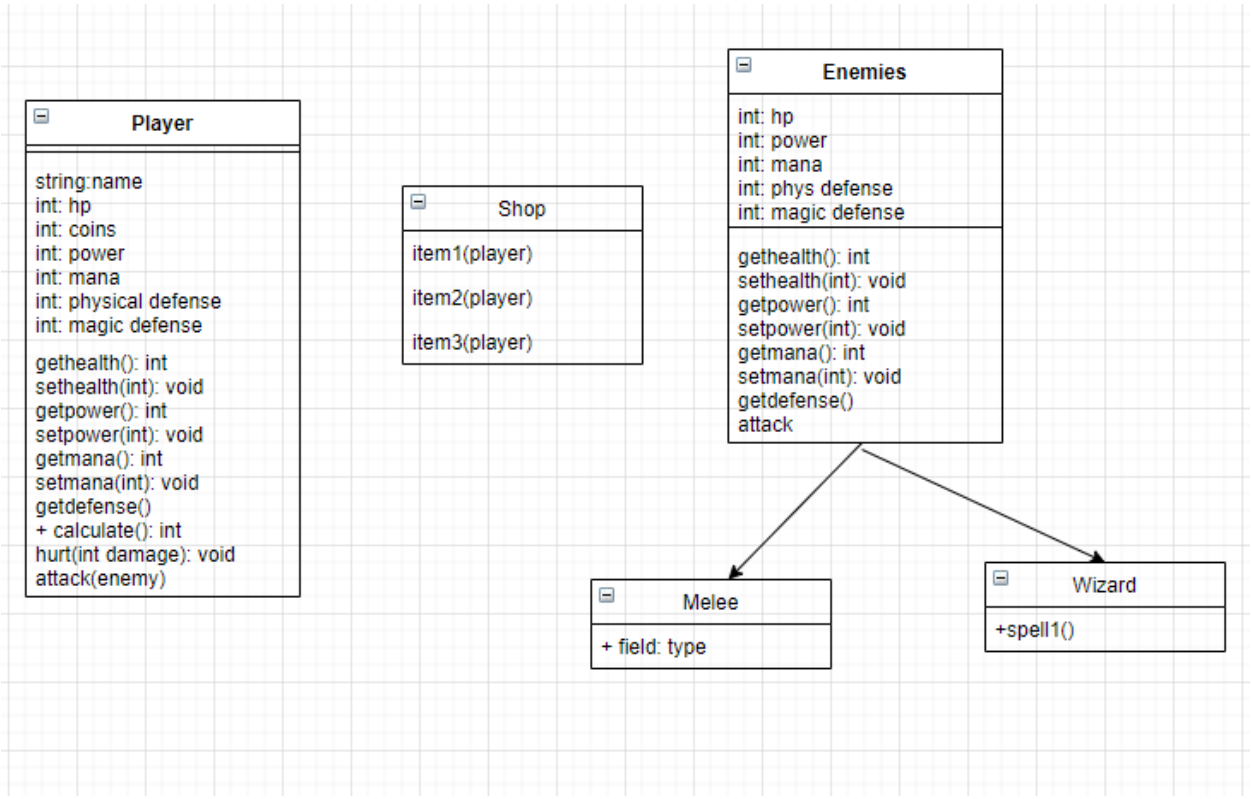
Final State of System Statement:

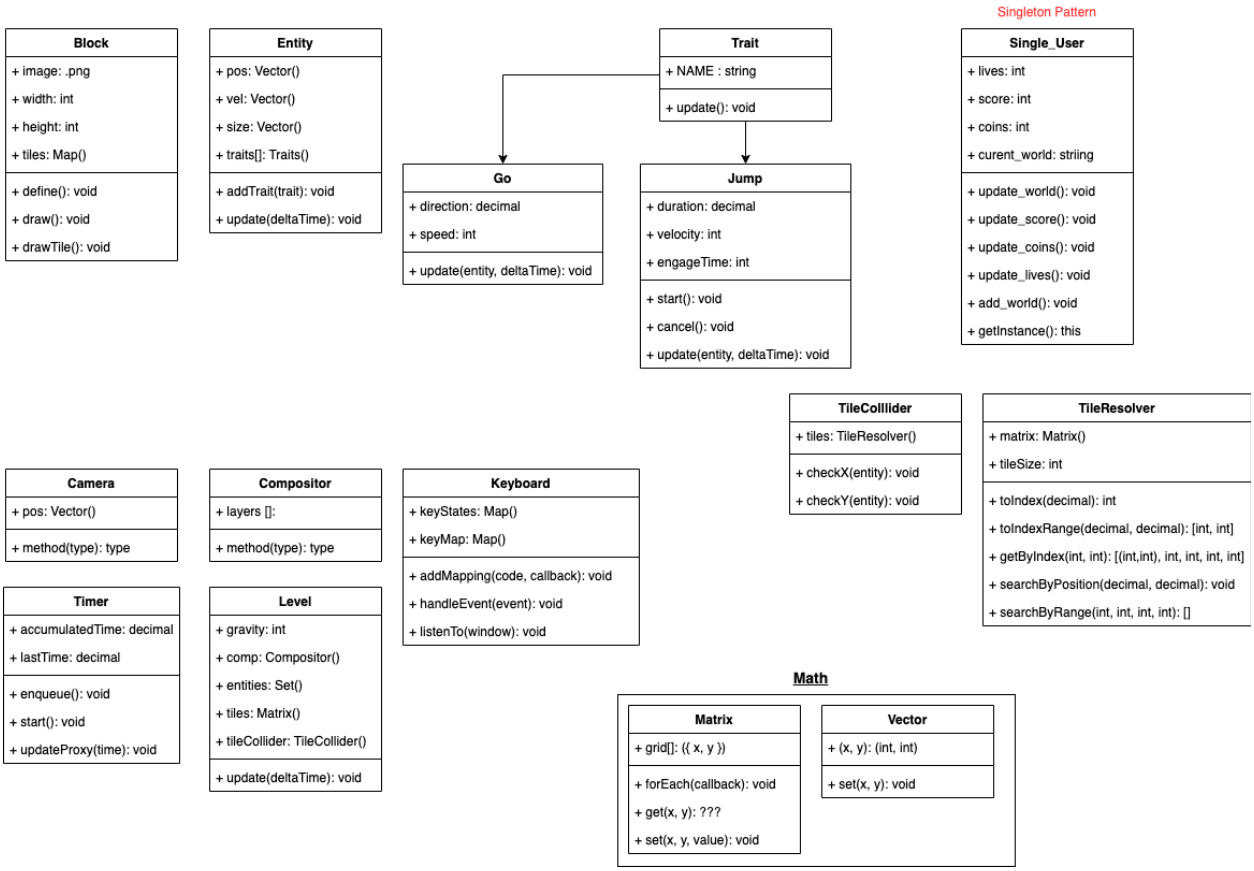
We were able to execute the website and put three games up onto the website. For the Super Ralphie Bros game, one level was created (the goal was 2), Ralphie was able to move and detect collisions with the blocks in the game. However, super powers were not yet added to the game and although certain attributes are shown in the console, not all the visual/graphics were able to be implemented for each action of the game.

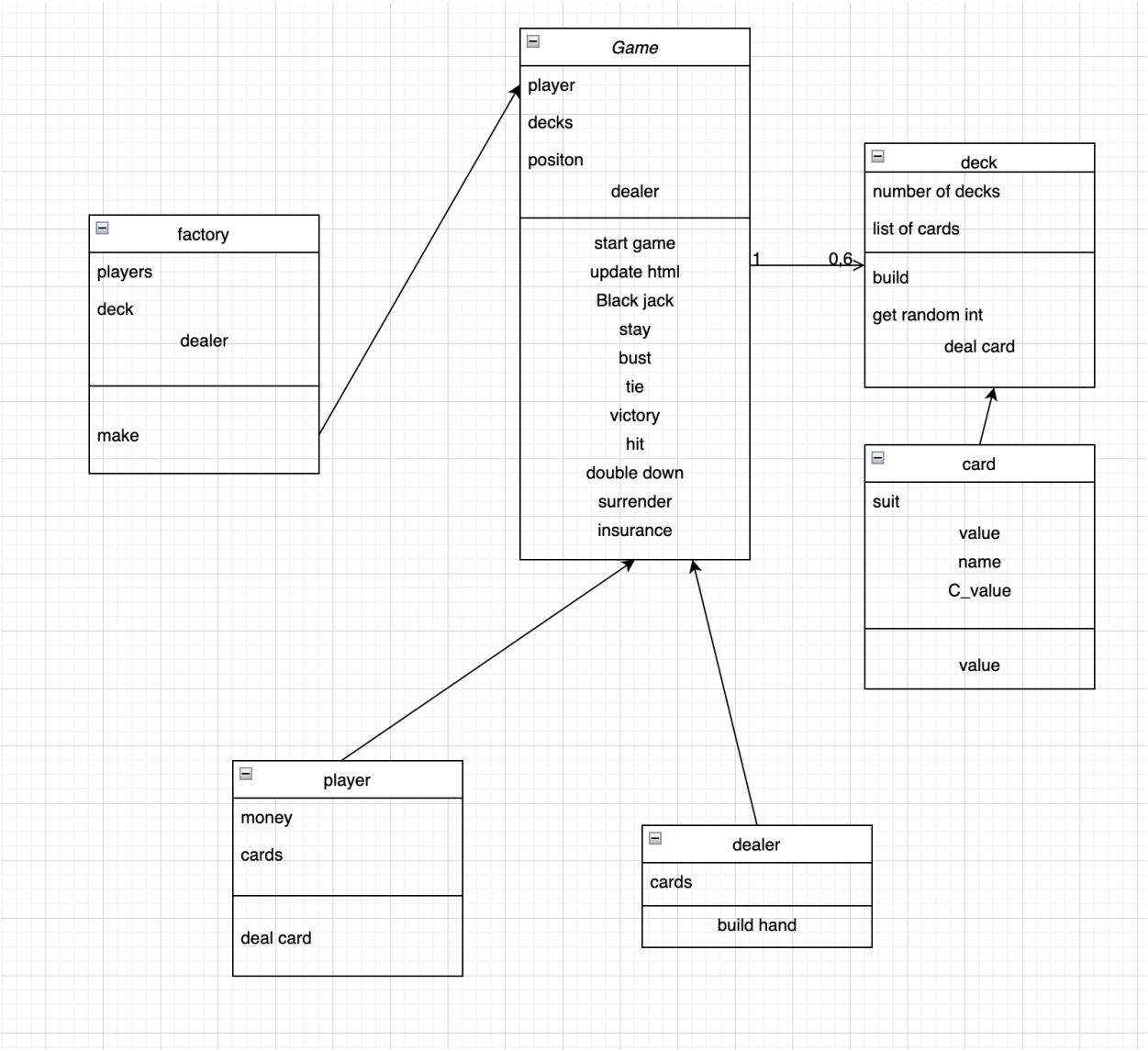
For Card counting black jack all the features were added except split money and multiple players. These were not added because the HTML for allowing these operations was outside of my skill level. These operations could have been implemented in the javascript easily .

Class Diagram

The changes from part 5 to part 7 were quite significant. Pretty much everything changes onced I learned how to work with the <canvas> html tag and how to render graphics in this tag.

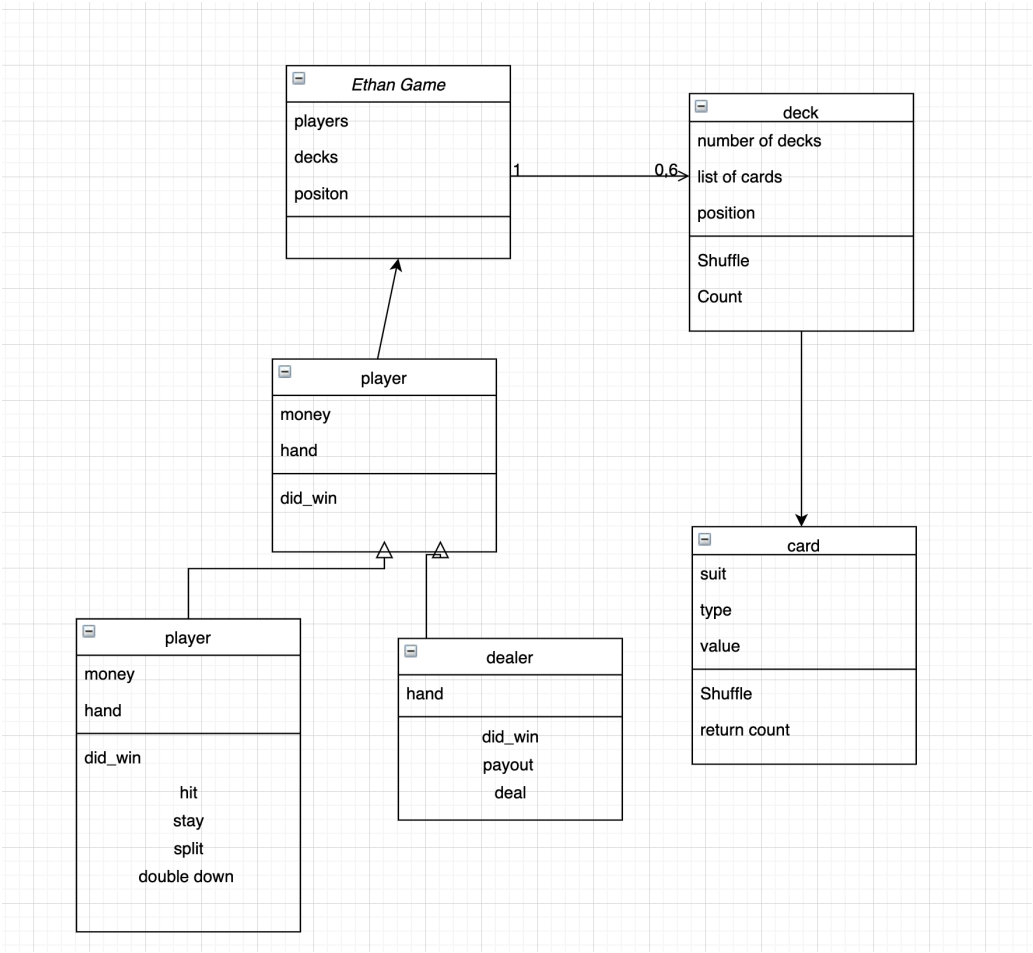

Player
string:name
int: hp
int: coins
int: power
int: mana
int: physical defense
int: magic defense
gethealth(): int
sethealth(int): void
getpower(): int
setpower(int): void
getmana(): int
setmana(int): void
getdefense()
+ calculate(): int
hurt(int damage): void
attack(enemy)
Shop
item1(player)
item2(player)
item3(player)
Enemies
int: hp
int: power
int: mana
int: phys defense
int: magic defense
gethealth(): int
sethealth(int): void
getpower(): int
setpower(int): void
getmana(): int
setmana(int): void
getdefense()
attack
Melee
+ field: type
Wizard
+spell1()


Block
+ image: .png
+ width: int
+ height: int
+ tiles: Map()
+ define(): void
+ draw(): void
+ drawTile(): void
Entity
+ pos: Vector()
+ vel: Vector()
+ size: Vector()
+ traits[]: Traits()
+ addTrait(trait): void
+ update(deltaTime): void
Trait
+ NAME : string
+ update(): void
Go
+ direction: decimal
+ speed: int
+ update(entity, deltaTime): void
Jump
+ duration: decimal
+ velocity: int
+ engageTime: int
+ start(): void
+ cancel(): void
+ update(entity, deltaTime): void
Singleton Pattern
Single_User
+ lives: int
+ score: int
+ coins: int
+ curent_world: striing
+ update_world(): void
+ update_score(): void
+ update_coins(): void
+ update_lives(): void
+ add_world(): void
+ getInstance(): this
TileColllider
+ tiles: TileResolver()
+ checkX(entity): void
+ checkY(entity): void
TileResolver
+ matrix: Matrix()
+ tileSize: int
+ toIndex(decimal): int
+ toIndexRange(decimal, decimal): [int, int]
+ getByIndex(int, int): [(int,int), int, int, int, int]
+ searchByPosition(decimal, decimal): void
+ searchByRange(int, int, int, int): []
Camera
+ pos: Vector()
+ method(type): type
Compositor
+ layers []:
+ method(type): type
Keyboard
+ keyStates: Map()
+ keyMap: Map()
+ addMapping(code, callback): void
+ handleEvent(event): void
+ listenTo(window): void
Timer
+ accumulatedTime: decimal
+ lastTime: decimal
+ enqueue(): void
+ start(): void
+ updateProxy(time): void
Level
+ gravity: int
+ comp: Compositor()
+ entities: Set()
+ tiles: Matrix()
+ tileCollider: TileCollider()
+ update(deltaTime): void
Math
Matrix
+ grid[]: ({ x, y })
+ forEach(callback): void
+ get(x, y): ???
+ set(x, y, value): void
Vector
+ (x, y): (int, int)
+ set(x, y): void



The Patterns I used were factories for implementing my players,dealers,and decks inside of the start game and facade to deal with cards. I chose these because implementing all my classes was not as clean as I would like and I wanted a place to do it and facade because it made dealing cards much easier.

Project 5 uml:



The key changes I made were making the player and dealer different classes and I moved a lot of the functions to the game class.I changed the position of my functions to save on code duplication and make the game flow more smoothly. I also implemented a factory to declare all my classes.

Third-Party Code

For the navbar used on the website we used code from BrianCodex, who has both a youtube and github page under this name.

For the Super Mario game, we used help from several youtube videos to help with the movement of the idle in the game and also for “Tile Collision”. The link below is the main video we used for these attributes of the game.
https://www.youtube.com/watch?v=YLMP5jmtpYc&ab_channel=MethMethMethod

For the Card counting black jack the HTML and CSS was inspired by
https://codepen.io/eqmvii/pen/GmRmZa

Some of the logic was inspired by the website as well.

OOAD Process

1. One design element we struggled with was being able to identify and pick which patterns would be able to fit best within our game. Sometimes, it felt best to not use a pattern at all, so deciding when to use them or when to not use patterns was a struggle. However, in the situations where we did end up picking patterns, they fit in very nicely with the rest of the code and it was easy to edit the classes and functions that were using clear patterns.
2. Another issue that we had to resolve was that we picked Javascript as our programming language. The way patterns are implemented in java are not implemented exactly the same in javascript, so we had to be able to convert the patterns we had learned to be able to work the same in a different language.
3. Having to brainstorm before writing code was very useful. I found myself referencing back to my UML diagram during the development process and it allowed me to not get lost in the code. This will definitely be my new process for big programs because it allows you to see issues before you have to make some strange code.