Project Part 4: Final Report

1. What features were implemented and a class diagram showing the final set of classes and relationships of the system. *(This may have changed from what you originally anticipated from earlier submissions.)* Discuss what changed in your class diagram and why it changed, or how it helped doing the diagrams first before coding if you did not need to change much. **From the discussion board clarification for Number 1: List the features you implemented (from part 1 and 2 where you listed the features you were planning to implement/requirements you listed).**

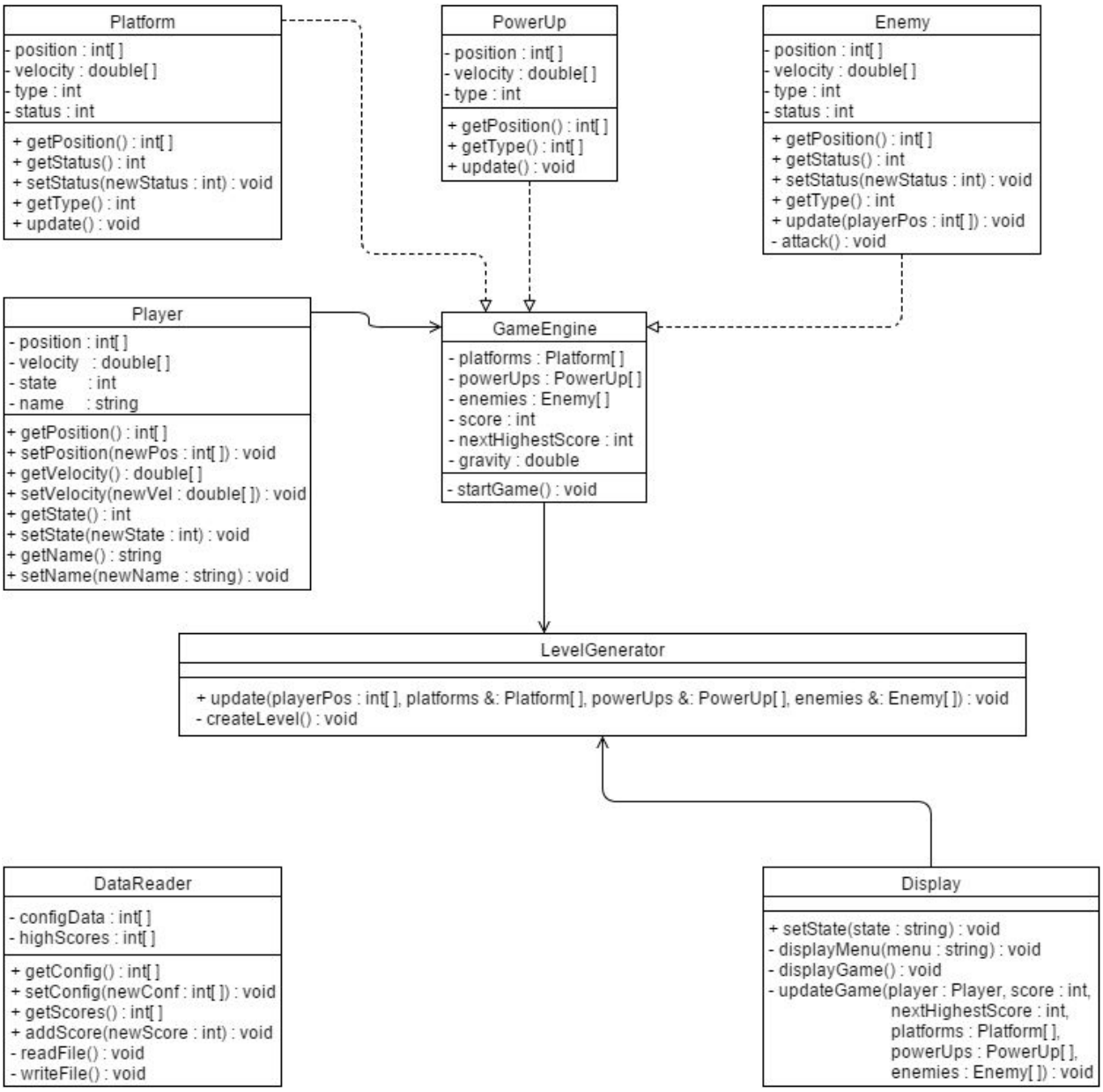
   We have implemented pretty much all the basics that we had planned. We have a class handling user input, and a class which randomly generates platforms for the levels. Further, we have our screen set to refresh at 60 frames per second, so as long as the user's computer is remotely decent, the lag between input and system reaction should be minimal. Unfortunately we did not implement a pause game option, however we did implement an unplanned game over menu which allows users to reenter the game quickly. We figure these two requirements are pretty "tit for tat" as if a user dies because they cannot pause, they can quickly get another game going. We did implement a leaderboard, however we did not implement a settings menu, because during development we realized there were not really any settings we thought the user would need to mess with.

2. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

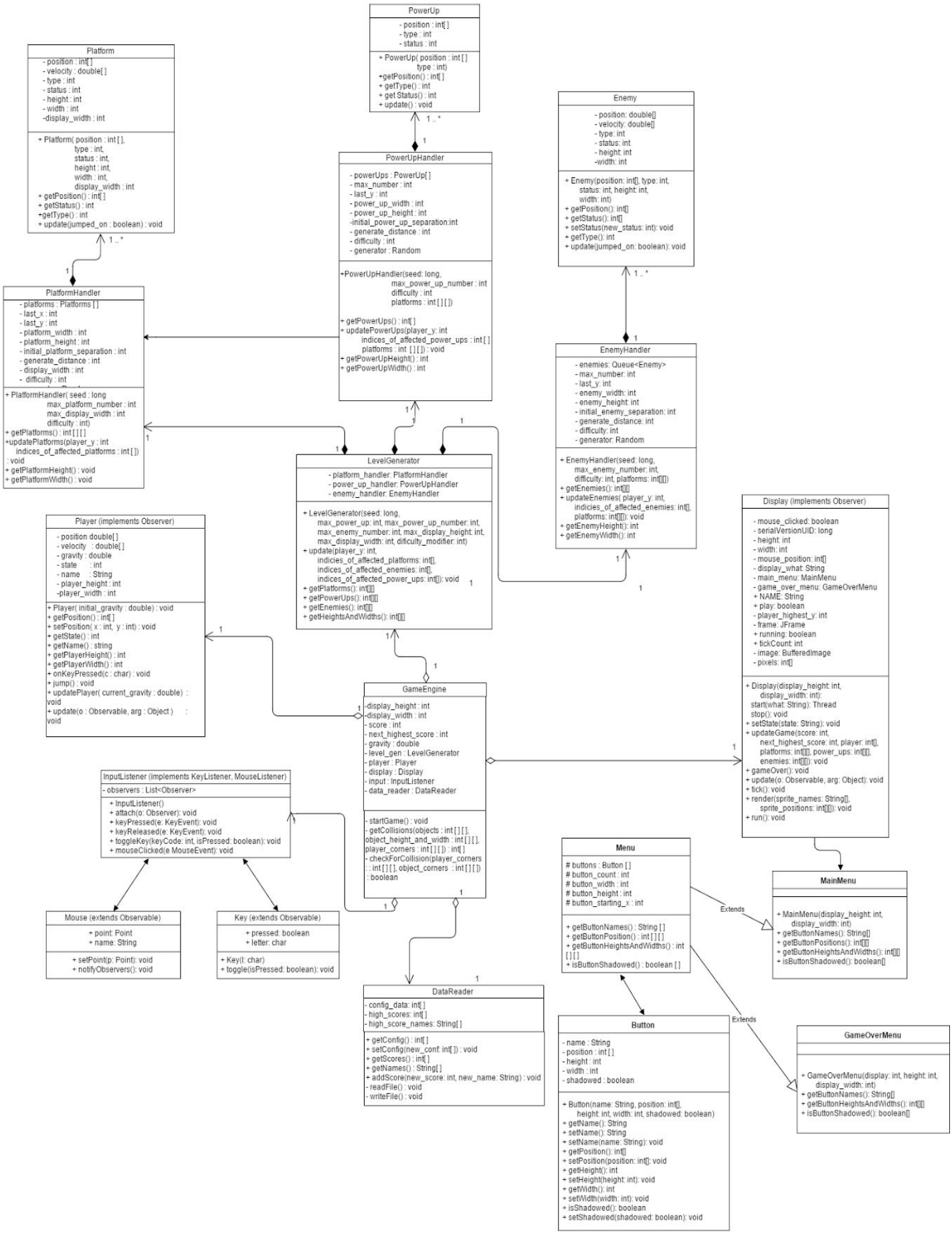
   In our final prototype, the main design pattern we used was the Observer pattern. Our observables are Key objects and Mouse objects, which live in an overall handler called inputListener. The observers are the Player and Display classes. This design pattern allows inputListener to take input from the user and pass it to the appropriate Key or Mouse object, which then, in turn, notifies Player and Display of the change of state. Player and Display then change depending on what key and mouse buttons were pressed. One design pattern we could have made use of was Factory. We had it in mind when making our Enemies, Platforms and PowerUps, but I would not say we actually used the pattern.

3. In addition, the report must discuss how the final system changed from the design you presented in Project Part 2. IN particular, include the class diagram you submitted for Project Part 2 and use it to compare and contrast with the class diagram that represents the final state of the system. **From the discussion board clarification for Number 3: Compare your Part 2 class diagram and your final class diagram - include part 2 class diagram and point out the necessary changes. You can answer why you didn't realize you needed things during the design and/or how next time you will be able to make a better design after this learning experience**

Original Class Diagram:



Final Class Diagram:


PowerUp
- position : int[ ]
- type : int
- status : int
+ PowerUp(position : int [ ]
type : int)
+getPosition() : int[ ]
+ getType() : int
+ get Status() : int
+ update() : void
1 .. *
1
Platform
- position : int[ ]
- velocity : double[ ]
- type : int
- status : int
- height : int
- width : int
-display_width : int
+ Platform(position : int [ ],
type : int,
status : int,
height : int,
width : int,
display_width : int
+ getPosition() : int[ ]
+ getStatus() : int
+getType() : int
+ update(jumped_on : boolean) : void
1 .. *
1
PowerUpHandler
- powerUps : PowerUp[ ]
- max_number : int
- last_y : int
- power_up_width : int
- power_up_height : int
-inital_power_up_separation:int
- generate_distance : int
- difficulty : int
- generator : Random
+PowerUpHandler(seed: long,
max_power_up_number : int
difficulty : int
platforms : int [ ] [ ])
+ getPowerUps() : int[ ]
+ updatePowerUps(player_y: int
indices_of_affected_power_ups : int [ ]
platforms : int [ ] [ ]) : void
+ getPowerUpHeight() : int
+ getPowerUpWidth() : int
Enemy
- position: double[]
- velocity: double[]
- type: int
- status: int
- height: int
-width: int
+ Enemy(position: int[], type: int,
status: int, height: int,
width: int)
+ getPosition(): int[]
+ getStatus(): int[]
+ setStatus(new_status: int): void
+ getType(): int
+ update(jumped_on: boolean): void
1 .. *
1
PlatformHandler
- platforms : Platforms [ ]
- last_x : int
- last_y : int
- platform_width : int
- platform_height : int
- initial_platform_separation : int
- generate_distance : int
- display_width : int
- difficulty : int
+ PlatformHandler(seed : long
max_platform_number : int
max_display_width : int
difficulty : int)
+ getPlatforms() : int [ ] [ ]
+updatePlatforms(player_y : int
indices_of_affected_platforms : int [ ])
: void
+ getPlatformHeight() : void
+ getPlatformWidth() : void
EnemyHandler
- enemies: Queue<Enemy>
- max_number: int
- last_y: int
- enemy_width: int
- enemy_height: int
- initial_enemy_separation: int
- generate_distance: int
- difficulty: int
- generator: Random
+ EnemyHandler(seed: long,
max_enemy_number: int,
difficulty: int, platforms: int[][])
+ getEnemies(): int[][]
+ updateEnemies(player_y: int,
indicies_of_affected_enemies: int[],
platforms: int[][]): void
+ getEnemyHeight(): int
+ getEnemyWidth(): int
LevelGenerator
- platform_handler: PlatformHandler
- power_up_handler: PowerUpHandler
- enemy_handler: EnemyHandler
+ LevelGenerator(seed: long,
max_power_up: int, max_power_up_number: int,
max_enemy_number: int, max_display_height: int,
max_display_width: int, dificulty_modifier: int)
+ update(player_y: int,
indicies_of_affected_platforms: int[],
indices_of_affected_enemies: int[],
indices_of_affected_power_ups: int[]): void
+ getPlatforms(): int[][]
+ getPowerUps(): int[][]
+ getEnemies(): int[][]
+ getHeightsAndWidths(): int[][]
Player (implements Observer)
- position double[ ]
- velocity : double[ ]
- gravity : double
- state : int
- name : String
- player_height : int
-player_width : int
+ Player(initial_gravity : double) : void
+ getPosition() : int[ ]
+ setPosition(x : int, y : int) : void
+ getState() : int
+ getName() : string
+ getPlayerHeight() : int
+ getPlayerWidth() : int
+ onKeyPressed(c : char) : void
+ jump() : void
+ updatePlayer(current_gravity : double) :
void
+ update(o : Observable, arg : Object) :
void
Display (implements Observer)
- mouse_clicked: boolean
- serialVersionUID: long
- height: int
- width: int
- mouse_position: int[]
- display_what: String
- main_menu: MainMenu
- game_over_menu: GameOverMenu
+ NAME: String
+ play: boolean
- player_highest_y: int
- frame: JFrame
+ running: boolean
+ tickCount: int
- image: BufferedImage
- pixels: int[]
+ Display(display_height: int,
display_width: int):
start(what: String): Thread
stop(): void
+ setState(state: String): void
+ updateGame(score: int,
next_highest_score: int, player: int[],
platforms: int[][], power_ups: int[][],
enemies: int[][]): void
+ gameOver(): void
+ update(o: Observable, arg: Object): void
+ tick(): void
+ render(sprite_names: String[],
sprite_positions: int[][]): void
+ run(): void
GameEngine
-display_height : int
-display_width : int
- score : int
- next_highest_score : int
- gravity : double
- level_gen : LevelGenerator
- player : Player
- display : Display
- input : InputListener
- data_reader : DataReader
- startGame() : void
- getCollisions(objects : int [ ] [ ],
object_height_and_width : int [ ] [ ],
player_corners : int [ ] [ ]) : int[ ]
- checkForCollision(player_corners
: int [ ] [ ], object_corners : int [ ] [ ])
: boolean
InputListener (implements KeyListener, MouseListener)
- observers : List<Observer>
+ InputListener()
+ attach(o: Observer): void
+ keyPressed(e: KeyEvent): void
+ keyReleased(e: KeyEvent): void
+ toggleKey(keyCode: int, isPressed: boolean): void
+ mouseClicked(e MouseEvent): void
Mouse (extends Observable)
+ point: Point
+ name: String
+ setPoint(p: Point): void
+ notifyObservers(): void
Key (extends Observable)
+ pressed: boolean
+ letter: char
+ Key(l: char)
+ toggle(isPressed: boolean): void
DataReader
- config_data: int[ ]
- high_scores: int[ ]
- high_score_names: String[ ]
+ getConfig() : int[ ]
+ setConfig(new_conf: int[ ]) : void
+ getScores() : int[ ]
+ getNames() : String[ ]
+ addScore(new_score: int, new_name: String) : void
- readFile() : void
- writeFile() : void
Menu
buttons : Button [ ]
button_count : int
button_width : int
button_height : int
button_starting_x : int
+ getButtonNames() : String [ ]
+ getButtonPosition() : int [ ] [ ]
+ getButtonHeightsAndWidths() : int
[ ] [ ]
+ isButtonShadowed() : boolean [ ]
Extends
MainMenu
+ MainMenu(display_height: int,
display_width: int)
+ getButtonNames(): String[]
+ getButtonPositions(): int[][]
+ getButtonHeightsAndWidths(): int[][]
+ isButtonShadowed(): boolean[]
Button
- name : String
- position : int [ ]
- height : int
- width : int
- shadowed : boolean
+ Button(name: String, position: int[],
height: int, width: int, shadowed: boolean)
+ getName(): String
+ setName(): String
+ setName(name: String): void
+ getPosition(): int[]
+ setPosition(position: int[]: void
+ getHeight(): int
+ setHeight(height: int): void
+ getWidth(): int
+ setWidth(width: int): void
+ isShadowed(): boolean
+ setShadowed(shadowed: boolean): void
Extends
GameOverMenu
+ GameOverMenu(display: int, height: int,
display_width: int)
+ getButtonNames(): String[]
+ getButtonHeightsAndWidths(): int[][]
+ isButtonShadowed(): boolean[]
1
1 .. *

3 cont) **The difference between our final and our initial class diagram is absolutely staggering. Initially we believed that our game was going to require about 8 classes. As of the final submission it has 18, more than double what we anticipated. Three of those new classes came in the form of "handlers" for our games main components (Power ups, enemies and platforms). The handlers purpose was to allow the level generator an easy way to get all of these component's data. We also added three classes which were all concerned with handling user input. We added four additional classes to handle the main menu and button selection process. Realistically our big takeaway was that we underestimated the complexity of the system we would need to implement this game. Some of these classes were made in order to keep our classes moderately sized and easy to read (like the handler classes), but we still ended up needing to implement quite a bit more than anticipated. Therefore, we definitely learned that REALLY thinking through that system analysis step can be a great move. When it came time to actually start coding it became apparent that we didn't properly analyze our needs during that step. Next time I think we all know that that process should be taken even more seriously than we initially thought.**

4. **From the discussion board clarification for Number 4: What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?**

   We learned that the process of analysis and design is critical to having a professional system/project. Further, we learned that the analysis process can be incredibly helpful when it comes to the design and implementation phases. We saw first hand why the diagrams in project part 2 were so emphasised; it became obvious that thinking through those diagrams and the system requirements can lead to a much easier implementation phase. In our case we underestimated what our system would need initially, but were able to use design principles from the course to implement the classes we needed to add. In the future, we will know to spend even more time analyzing our requirements, in order to avoid the underestimating problem again. Overall, we all took away an appreciation for the early analysis phases of a project, as putting in hard work in that arena can lead to much smoother coding down the road. I think we all came from a background where you just kind of "dive in" when it comes to coding. After this class I am confident that all group members will likely go through the analysis and design phase, at least to some extent, before "diving in" to their next coding project.