Project Final Report

Team

Bradley Arnot
Kelsey Dowd
Jeffrey Lipnick
Nicholas Pfeufer

Title

Santa Horror RPG

List the features that were implemented (table with ID and title)

Use Case ID	Use Case Name
UC-01	Play Game (partially implemented)
UC-03	Load Save File
UC-04	Move Character
UC-07	Quit Game

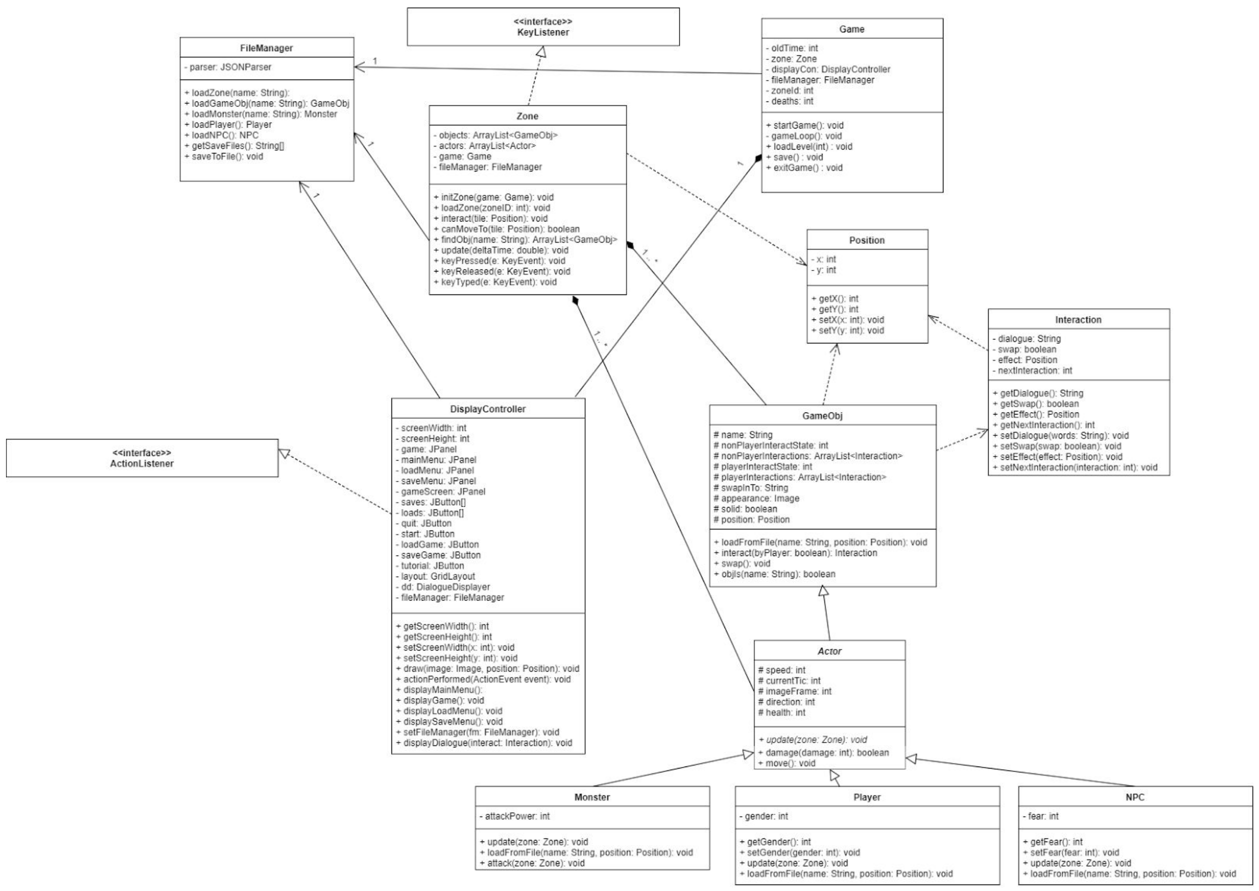
List the features were not implemented from Part 2 (table with ID and title)

Use Case ID	Use Case Name
UC-02	Save Game
UC-05	View Game Statistics
UC-06	Solve Puzzles
UC-08	View Tutorial
UC-09	Interact with Objects
UC-10	Fight Bosses

Show your Part 2 class diagram and your final class diagram

Part 2 Class Diagram

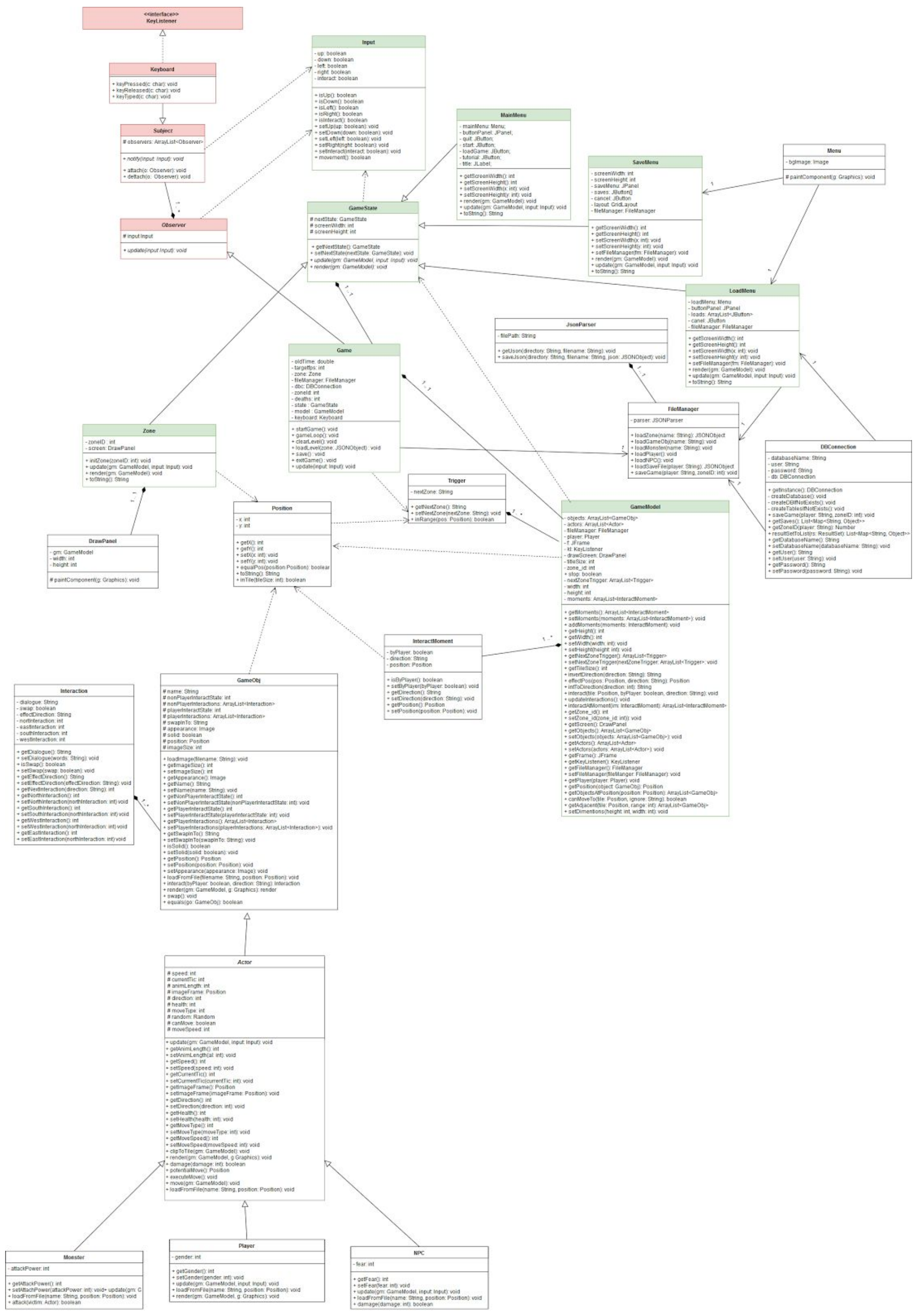
Diagram can be viewed in more detail on GitHub at “Diagrams/ClassDiagram.[png/svg]”



<<interface>> KeyListener
FileManager
- parser: JSONParser
+ loadZone(name: String):
+ loadGameObj(name: String): GameObj
+ loadMonster(name: String): Monster
+ loadPlayer(): Player
+ loadNPC(): NPC
+ getSaveFiles(): String[]
+ saveToFile(): void
Game
- oldTime: int
- zone: Zone
- displayCon: DisplayController
- fileManager: FileManager
- zoneId: int
- deaths: int
+ startGame(): void
- gameLoop(): void
+ loadLevel(int) : void
+ save() : void
+ exitGame() : void
Zone
- objects: ArrayList<GameObj>
- actors: ArrayList<Actor>
- game: Game
- fileManager: FileManager
+ initZone(game: Game): void
+ loadZone(zoneID: int): void
+ interact(tile: Position): void
+ canMoveTo(tile: Position): boolean
+ findObj(name: String): ArrayList<GameObj>
+ update(deltaTime: double): void
+ keyPressed(e: KeyEvent): void
+ keyReleased(e: KeyEvent): void
+ keyTyped(e: KeyEvent): void
Position
- x: int
- y: int
+ getX(): int
+ getY(): int
+ setX(x: int): void
+ setY(y: int): void
Interaction
- dialogue: String
- swap: boolean
- effect: Position
- nextInteraction: int
+ getDialogue(): String
+ getSwap(): boolean
+ getEffect(): Position
+ getNextInteraction(): int
+ setDialogue(words: String): void
+ setSwap(swap: boolean): void
+ setEffect(effect: Position): void
+ setNextInteraction(interaction: int): void
<<interface>> ActionListener
DisplayController
- screenWidth: int
- screenHeight: int
- game: JPanel
- mainMenu: JPanel
- loadMenu: JPanel
- saveMenu: JPanel
- gameScreen: JPanel
- saves: JButton[]
- loads: JButton[]
- quit: JButton
- start: JButton
- loadGame: JButton
- saveGame: JButton
- tutorial: JButton
- layout: GridLayout
- dd: DialogueDisplayer
- fileManager: FileManager
+ getScreenWidth(): int
+ getScreenHeight(): int
+ setScreenWidth(x: int): void
+ setScreenHeight(y: int): void
+ draw(image: Image, position: Position): void
+ actionPerformed(ActionEvent event): void
+ displayMainMenu():
+ displayGame(): void
+ displayLoadMenu(): void
+ displaySaveMenu(): void
+ setFileManager(fm: FileManager): void
+ displayDialogue(interact: Interaction): void
GameObj
# name: String
# nonPlayerInteractState: int
# nonPlayerInteractions: ArrayList<Interaction>
# playerInteractState: int
# playerInteractions: ArrayList<Interaction>
# swapInTo: String
# appearance: Image
# solid: boolean
# position: Position
+ loadFromFile(name: String, position: Position): void
+ interact(byPlayer: boolean): Interaction
+ swap(): void
+ objIs(name: String): boolean
Actor
# speed: int
# currentTic: int
# imageFrame: int
# direction: int
# health: int
+ update(zone: Zone): void
+ damage(damage: int): boolean
+ move(): void
Monster
- attackPower: int
+ update(zone: Zone): void
+ loadFromFile(name: String, position: Position): void
+ attack(zone: Zone): void
Player
- gender: int
+ getGender(): int
+ setGender(gender: int): void
+ update(zone: Zone): void
+ loadFromFile(name: String, position: Position): void
NPC
- fear: int
+ getFear(): int
+ setFear(fear: int): void
+ update(zone: Zone): void
+ loadFromFile(name: String, position: Position): void
1
1
1
1
1..*
1..*

Final Class Diagram

Diagram can be viewed in more detail on GitHub at "Diagrams/FINALClassDiagram.[png/svg]"



<<interface>>
KeyListener
Keyboard
+ keyPressed(c: char): void
+ keyReleased(c: char): void
+ keyTyped(c: char): void
Subject
# observers: ArrayList<Observer>
+ notify(input: Input): void
+ attach(o: Observer): void
+ deltach(o: Observer): void
Observer
# input Input
+ update(input Input): void
Input
- up: boolean
- down: boolean
- left: boolean
- right: boolean
- interact: boolean
+ isUp(): boolean
+ isDown(): boolean
+ isLeft(): boolean
+ isRight(): boolean
+ isInteract(): boolean
+ setUp(up: boolean): void
+ setDown(down: boolean): void
+ setLeft(left: boolean): void
+ setRight(right: boolean): void
+ setInteract(interact: boolean): void
+ movement(): boolean
GameState
# nextState: GameState
# screenWidth: int
# screenHeight: int
+ getNextState(): GameState
+ setNextState(nextState: GameState): void
+ update(gm: GameModel, input: Input): void
+ render(gm: GameModel): void
MainMenu
- mainMenu: Menu;
- buttonPanel: JPanel;
- quit: JButton;
- start: JButton;
- loadGame: JButton;
- tutorial: JButton;
- title: JLabel;
+ getScreenWidth(): int
+ getScreenHeight(): int
+ setScreenWidth(x: int): void
+ setScreenHeight(y: int): void
+ render(gm: GameModel): void
+ update(gm: GameModel, input: Input): void
+ toString(): String
SaveMenu
- screenWidth: int
- screenHeight: int
- saveMenu: JPanel
- saves: JButton[]
- cancel: JButton
- layout: GridLayout
- fileManager: FileManager
+ getScreenWidth(): int
+ getScreenHeight(): int
+ setScreenWidth(x: int): void
+ setScreenHeight(y: int): void
+ setFileManager(fm: FileManager): void
+ render(gm: GameModel): void
+ update(gm: GameModel, input: Input): void
+ toString(): String
Menu
- bgImage: Image
# paintComponent(g: Graphics): void
LoadMenu
- loadMenu: Menu
- buttonPanel: JPanel
- loads: ArrayList<JButton>
- canel: JButton
- fileManager: FileManager
+ getScreenWidth(): int
+ getScreenHeight(): int
+ setScreenWidth(x: int): void
+ setScreenHeight(y: int): void
+ setFileManager(fm: FileManager): void
+ render(gm: GameModel): void
+ update(gm: GameModel, input: Input): void
+ toString(): String
JsonParser
- filePath: String
+ getJson(directory: String, filename: String): void
+ saveJson(directory: String, filename: String, json: JSONObject): void
FileManager
- parser: JSONParser
+ loadZone(name: String): JSONObject
+ loadGameObj(name: String): void
+ loadMonster(name: String): void
+ loadPlayer(): void
+ loadNPC(): void
+ loadSaveFile(player: String): JSONObject
+ saveGame(player: String, zoneID: int): void
DBConnection
- databaseName: String
- user: String
- password: String
- db: DBConnection
+ getInstance(): DBConnection
- createDatabase(): void
- createDBIfNotExists(): void
- createTablesIfNotExists(): void
+ saveGame(player: String, zoneID: int): void
+ getSaves(): List<Map<String, Object>>
+ getZoneID(player: String): Number
+ resultSetToList(rs: ResultSet): List<Map<String, Object>>
+ getDatabaseName(): String
+ setDatabaseName(databaseName: String): void
+ getUser(): String
+ setUser(user: String): void
+ getPassword(): String
+ setPassword(password: String): void
Game
- oldTime: double
- targetfps: int
- zone: Zone
- fileManager: FileManager
- dbc: DBConnection
- zoneId: int
- deaths: int
- state : GameState
- model : GameModel
- keyboard: Keyboard
+ startGame(): void
+ gameLoop(): void
+ clearLevel(): void
+ loadLevel(zone: JSONObject) : void
+ save() : void
+ exitGame() : void
+ update(input: Input): void
Zone
- zoneID : int
- screen: DrawPanel
+ initZone(zoneID: int): void
+ update(gm: GameModel, input: Input): void
+ render(gm: GameModel): void
+ toString(): String
DrawPanel
- gm: GameModel
- width: int
- height: int
# paintComponent(g: Graphics): void
Position
- x: int
- y: int
+ getX(): int
+ getY(): int
+ setX(x: int): void
+ setY(y: int): void
+ equalPos(position Position): boolear
+ toString(): String
+ inTile(tileSize: int): boolean
Trigger
- nextZone: String
+ getNextZone(): String
+ setNextZone(nextZone: String): void
+ inRange(pos: Position): boolean
GameModel
- objects: ArrayList<GameObj>
- actors: ArrayList<Actor>
- fileManager: FileManager
- player: Player
- f: JFrame
- kl: KeyListener
- drawScreen: DrawPanel
- tileSize: int
- zone_id: int
+ stop: boolean
- nextZoneTrigger: ArrayList<Trigger>
- width: int
- height: int
- moments: ArrayList<InteractMoment>
+ getMoments(): ArrayList<InteractMoment>
+ setMoments(moments: ArrayList<InteractMoment>): void
+ addMoments(moments: InteractMoment): void
+ getHeight(): int
+ getWidth(): int
+ setWidth(width: int): void
+ setHeight(height: int): void
+ getNextZoneTrigger(): ArrayList<Trigger>
+ setNextZoneTrigger(nextZoneTrigger: ArrayList<Trigger>: void
+ getTileSize(): int
+ invertDirection(direction: String): String
+ effectPos(pos: Position, direction: String): Position
+ intToDirection(direction: int): String
+ interact(tile: Position, byPlayer: boolean, direction: String): void
+ updateInteractions(): void
+ interactAtMoment(im: InteractMoment): ArrayList<InteractMoment>
+ getZone_id(): int
+ setZone_id(zone_id: int)): void
+ getScreen(): DrawPanel
+ getObjects(): ArrayList<GameObj>
+ setObjects(objects: ArrayList<GameObj>): void
+ getActors(): ArrayList<Actor>
+ setActors(actors: ArrayList<Actor>): void
+ getFrame(): JFrame
+ getKeyListener(): KeyListener
+ getFileManager(): FileManager
+ setFileManager(fileManger: FileManager): void
+ getPlayer(player: Player): void
+ getPosition(object: GameObj): Position
+ getObjectsAtPosition(position: Position): ArrayList<GameObj>
+ canMoveTo(tile: Position, ignore: String): boolean
+ getAdjacent(tile: Position, range: int): ArrayList<GameObj>
+ setDimentions(height: int, width: int): void
InteractMoment
- byPlayer: boolean
- direction: String
- position: Position
+ isByPlayer(): boolean
+ setByPlayer(byPlayer: boolean): void
+ getDirection(): String
+ setDirection(direction: String): void
+ getPosition(): Position
+ setPosition(position: Position): void
Interaction
- dialogue: String
- swap: boolean
- effectDirection: String
- nortInteraction: int
- eastInteraction: int
- southInteraction: int
- westInteraction: int
+ getDialogue(): String
+ setDialogue(words: String): void
+ isSwap(): boolean
+ setSwap(swap: boolean): void
+ getEffectDirection(): String
+ setEffectDirection(effectDirection: String): void
+ getNextInteraction(direction: String): int
+ getNorthInteraction(): int
+ setNorthInteraction(northInteraction: int) void
+ getSouthInteraction(): int
+ setSouthInteraction(northInteraction: int) void
+ getWestInteraction(): int
+ setWestInteraction(northInteraction: int) void
+ getEastInteraction(): int
+ setEastInteraction(northInteraction: int) void
GameObj
# name: String
# nonPlayerInteractState: int
# nonPlayerInteractions: ArrayList<Interaction>
# playerInteractState: int
# playerInteractions: ArrayList<Interaction>
# swapInTo: String
# appearance: Image
# solid: boolean
# position: Position
# imageSize: int
+ loadImage(filename: String): void
+ getImageSize(): int
+ setImageSize(): int
+ getAppearance(): Image
+ getName(): String
+ setName(name: String): void
+ getNonPlayerInteractState(): int
+ setNonPlayerInteractState(nonPlayerInteractState: int): void
+ getPlayerInteractState(): int
+ setPlayerInteractState(playerInteractState: int): void
+ getPlayerInteractions(): ArrayList<Interaction>
+ setPlayerInteractions(playerInteractions: ArrayList<Interaction>): void
+ getSwapInTo(): String
+ setSwapInTo(swapInTo: String): void
+ isSolid(): boolean
+ setSolid(solid: boolean): void
+ getPosition(): Position
+ setPosition(position: Position): void
+ setAppearance(appearance: Image): void
+ loadFromFile(filename: String, position: Position): void
+ interact(byPlayer: boolean, direction: String): Interaction
+ render(gm: GameModel, g: Graphics): render
+ swap(): void
+ equals(go: GameObj): boolean
Actor
# speed: int
# currentTic: int
# animLength: int
# imageFrame: Position
# direction: int
# health: int
# moveType: int
# random: Random
# canMove: boolean
# moveSpeed: int
+ update(gm: GameModel, input: Input): void
+ getAnimLength(): int
+ setAnimLength(al: int): void
+ getSpeed(): int
+ setSpeed(speed: int): void
+ getCurrentTic(): int
+ setCurrentTic(currentTic: int): void
+ getImageFrame(): Position
+ setImageFrame(imageFrame: Position): void
+ getDirection(): int
+ setDirection(direction: int): void
+ getHealth(): int
+ setHealth(health: int): void
+ getMoveType(): int
+ setMoveType(moveType: int): void
+ getMoveSpeed(): int
+ setMoveSpeed(moveSpeed: int): void
+ clipToTile(gm: GameModel): void
+ render(gm: GameModel, g Graphics): void
+ damage(damage: int): boolean
+ potentialMove(): Position
+ executeMove(): void
+ move(gm: GameModel): void
+ loadFromFile(name: String, position: Position): void
Monster
- attackPower: int
+ getAttackPower(): int
+ setAttachPower(attackPower: int): void+ update(gm: C
+ loadFromFile(name: String, position: Position): void
+ attack(victim: Actor): boolean
Player
- gender: int
+ getGender(): int
+ setGender(gender: int): void
+ update(gm: GameModel, input: Input): void
+ loadFromFile(name: String, position: Position): void
+ render(gm: GameModel, g: Graphics): void
NPC
- fear: int
+ getFear(): int
+ setFear(fear: int): void
+ update(gm: GameModel, input: Input): void
+ loadFromFile(name: String, position: Position): void
+ damage(damage: int): boolean

What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

We saw a number of ways we could implement design patterns to improve the structure of our game. The main changes we made to the class diagram where to enact these improvements. In particular we changed the user input to be handled with the Observer design pattern and the Menus/Game levels to be handled with the State design pattern.

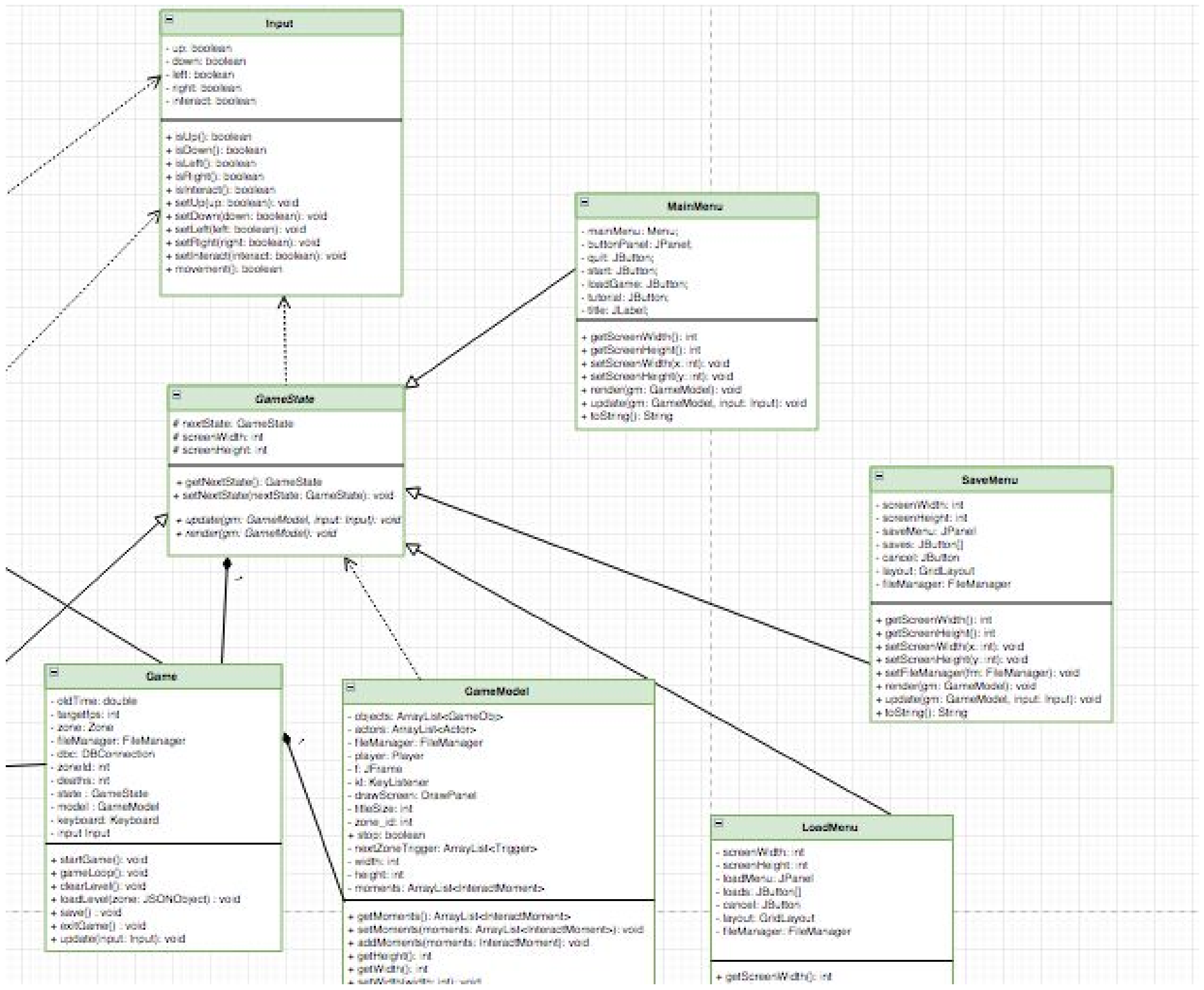
Did you make use of any design patterns in the implementation of your final Prototype? If so, how? Show the classes from your class diagram that implement each design pattern (each design pattern as a separate image in the .PDF).

Yes, we implemented both design patterns we had in our final prototype. The design patterns we implemented are listed on the following pages.

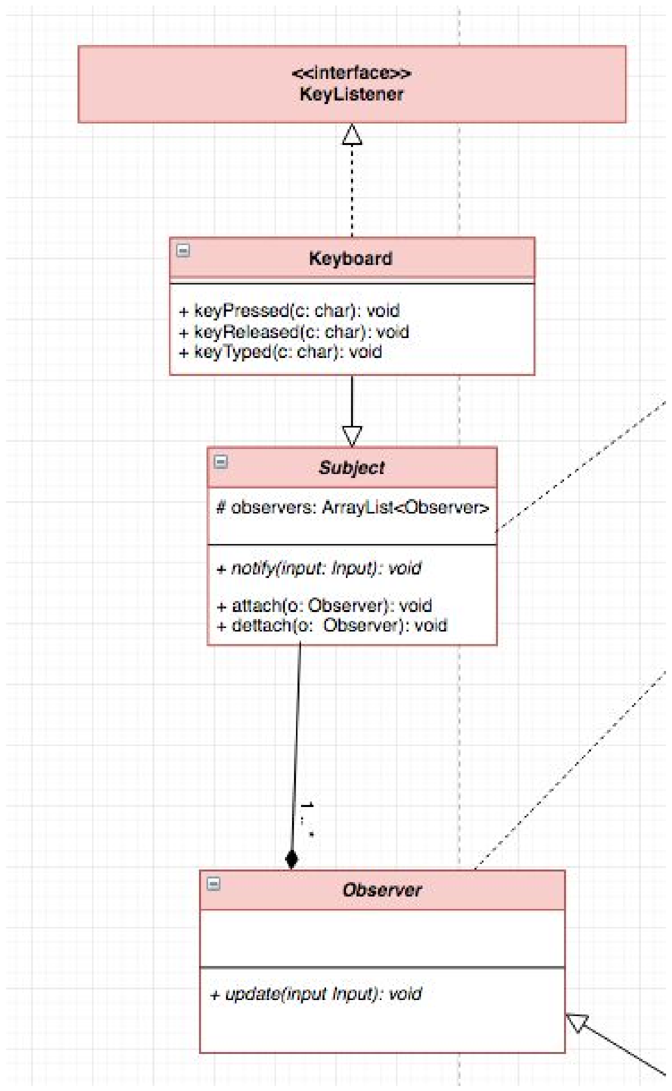
State Design Pattern

We refactored the structure of the game to use the State Design Pattern. These gave us a better format for loading the different menu states as well as the game state. Originally, most of this functionality was stored in the "Zone" class during Part 2, but during our refactor we saw this opportunity for improvement. Each menu and level in the game is an individual GameState set in the Game class. Each GameState determines the next state it will change to in the update function, this could be itself. Then Game changes to next the GameState. The current GameState also runs a render function every frame.

Diagram can be viewed in more detail on GitHub at "Diagrams/FINALClassDiagram.[png/svg]" The State design pattern classes are highlighted in Green.



Observer Design Pattern
To handle the player input we refactored the class diagram to use the Observer Design Pattern. We saw that the publish-subscribe was a much better way to handle input from different sources like a keyboard and mouse for extensibility. The Keyboard class inherits from KeyListener and gets the user keyboard input. It also extends the abstract Subject class so it has a list of Observers and a method to notify the Observers. The Observer that attaches to the Keyboard object is the Game object. Everytime a key is pressed, Keyboard notifies the Game object of any keyboard input.


<<interface>>
KeyListener
Keyboard
+ keyPressed(c: char): void
+ keyReleased(c: char): void
+ keyTyped(c: char): void
Subject
# observers: ArrayList<Observer>
+ notify(input: Input): void
+ attach(o: Observer): void
+ dettach(o: Observer): void
1..*
Observer
+ update(input Input): void

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?

Learning design patterns made a big difference in the overall structure of our game. It made various pieces like the menus and user input much easier than what we were originally planning. Additionally, using an iterative process was very helpful. After trying a certain design, we stopped and looked for ways to improve it using the concepts we learned in class.

Overall, we learned the value of using good OO design principles in implementing a system. If we were going to do this again, we would definitely start by looking at what design patterns would best fit the structure of our program and diagramming the system out. Also, on a more basic level, keeping ideas like scalability in mind is something that would drive our design.