Milestone Four: V2

Board Game Design Document

Student:

Lillith Chute & Donglin Xu

Teacher:

Maria Jump

Course:

CS 5010

Design Summary

This project is to design a game based very loosely on the Kill Doctor Lucky series of games. This project will be using the MVC pattern as its primary architecture. This second milestone concerned itself with adding a player and actions for that player to the game. Also, a computer controlled player was added that can perform the same actions as a human player. Additionally, a controller was added that utilized the command pattern.

The third milestone introduced a companion pet for the target. This pet has the special ability of making the contents of any room it is occupying invisible to the players. Thus, they cannot see inside that room. Players can move that pet to any room on the board as part of their turn whether they are in the same space as the pet or not. Also, players can now attack the target, so long as they are in the same room and can't be seen by any other player. Not being seen means no other players in the room nor in any neighboring room.

The game ends either because the target is dead or the maximum number of turns has been reached.

The fourth and final milestone for the project is to introduce a graphical user interface for the project. The project will continue to use the model as designed in the first three milestones. However, for this milestone we will no longer use the console based controller and associated collateral. Instead we will be creating a GUI using Java Swing components. All of the existing command behaviors will be present (pick up item, attack, and so on), they will just be executed through the GUI.

The expectation is that the requirements of the application will change over the course of the four milestones so the design will be flexible.

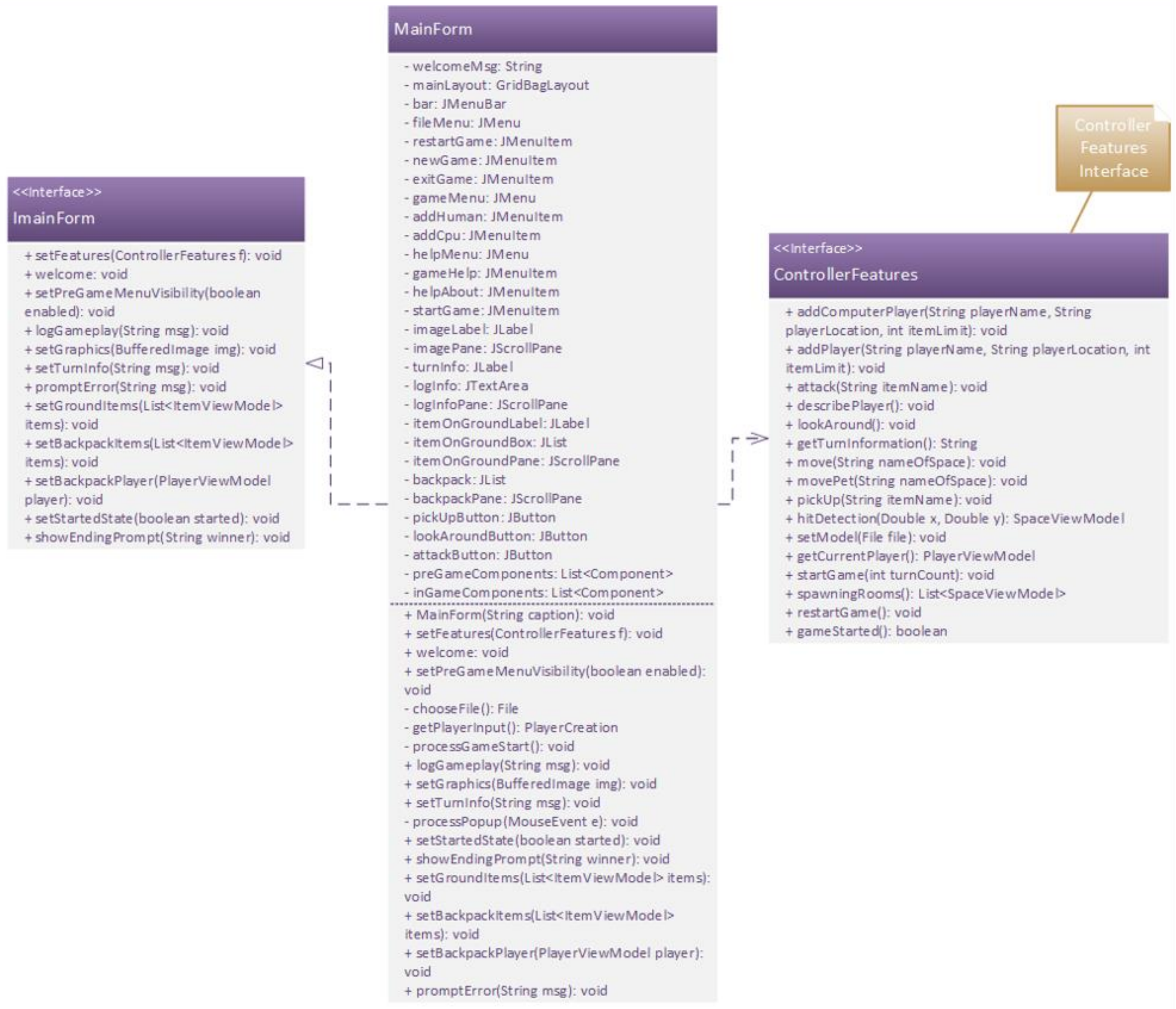
UML Diagram

Note: This design is utilizing the model as completed in milestone three. It creates view models to provide add separation of concern and read only properties for the view. It also continues the the use of a command pattern in conjunction with the controller. It also now adds on a view represented as a GUI.

Updated April 24, 2022

Milestone 4 Design Document

VIEW



<<Interface>>
ImainForm

+ setFeatures(ControllerFeatures f): void
+ welcome: void
+ setPreGameMenuVisibility(boolean enabled): void
+ logGameplay(String msg): void
+ setGraphics(BufferedImage img): void
+ setTurnInfo(String msg): void
+ promptError(String msg): void
+ setGroundItems(List<ItemViewModel> items): void
+ setBackpackItems(List<ItemViewModel> items): void
+ setBackpackPlayer(PlayerViewModel player): void
+ setStartedState(boolean started): void
+ showEndingPrompt(String winner): void

MainForm

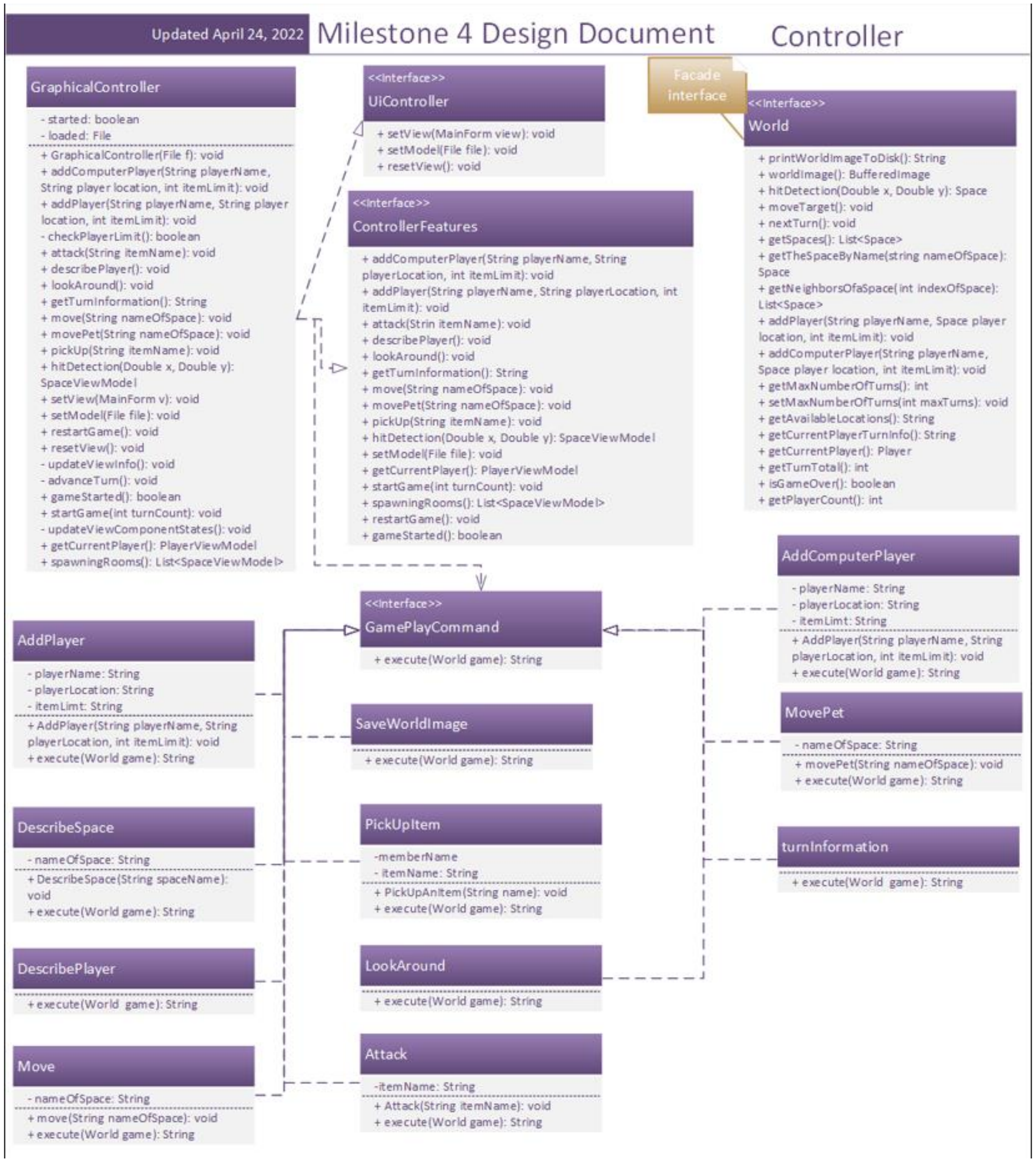
- welcomeMsg: String
- mainLayout: GridBagLayout
- bar: JMenuBar
- fileMenu: JMenu
- restartGame: JMenuItem
- newGame: JMenuItem
- exitGame: JMenuItem
- gameMenu: JMenu
- addHuman: JMenuItem
- addCpu: JMenuItem
- helpMenu: JMenu
- gameHelp: JMenuItem
- helpAbout: JMenuItem
- startGame: JMenuItem
- imageLabel: JLabel
- imagePane: JScrollPane
- turnInfo: JLabel
- logInfo: JTextArea
- logInfoPane: JScrollPane
- itemOnGroundLabel: JLabel
- itemOnGroundBox: JList
- itemOnGroundPane: JScrollPane
- backpack: JList
- backpackPane: JScrollPane
- pickUpButton: JButton
- lookAroundButton: JButton
- attackButton: JButton
- preGameComponents: List<Component>
- inGameComponents: List<Component>

+ MainForm(String caption): void
+ setFeatures(ControllerFeatures f): void
+ welcome: void
+ setPreGameMenuVisibility(boolean enabled): void
- chooseFile(): File
- getPlayerInput(): PlayerCreation
- processGameStart(): void
+ logGameplay(String msg): void
+ setGraphics(BufferedImage img): void
+ setTurnInfo(String msg): void
- processPopup(MouseEvent e): void
+ setStartedState(boolean started): void
+ showEndingPrompt(String winner): void
+ setGroundItems(List<ItemViewModel> items): void
+ setBackpackItems(List<ItemViewModel> items): void
+ setBackpackPlayer(PlayerViewModel player): void
+ promptError(String msg): void

Controller Features Interface

<<Interface>>
ControllerFeatures

+ addComputerPlayer(String playerName, String playerLocation, int itemLimit): void
+ addPlayer(String playerName, String playerLocation, int itemLimit): void
+ attack(String itemName): void
+ describePlayer(): void
+ lookAround(): void
+ getTurnInformation(): String
+ move(String nameOfSpace): void
+ movePet(String nameOfSpace): void
+ pickUp(String itemName): void
+ hitDetection(Double x, Double y): SpaceViewModel
+ setModel(File file): void
+ getCurrentPlayer(): PlayerViewModel
+ startGame(int turnCount): void
+ spawningRooms(): List<SpaceViewModel>
+ restartGame(): void
+ gameStarted(): boolean



Updated April 24, 2022 Milestone 4 Design Document Controller

```
GraphicalController
  - started: boolean
  - loaded: File
  + GraphicalController(File f): void
  + addComputerPlayer(String playerName, String player location, int itemLimit): void
  + addPlayer(String playerName, String player location, int itemLimit): void
  - checkPlayerLimit(): boolean
  + attack(String itemName): void
  + describePlayer(): void
  + lookAround(): void
  + getTurnInformation(): String
  + move(String nameOfSpace): void
  + movePet(String nameOfSpace): void
  + pickUp(String itemName): void
  + hitDetection(Double x, Double y): SpaceViewModel
  + setView(MainForm v): void
  + setModel(File file): void
  + restartGame(): void
  + resetView(): void
  - updateViewInfo(): void
  - advanceTurn(): void
  + gameStarted(): boolean
  + startGame(int turnCount): void
  - updateViewComponentStates(): void
  + getCurrentPlayer(): PlayerViewModel
  + spawningRooms(): List<SpaceViewModel>
```

```
<<Interface>>
UiController
  + setView(MainForm view): void
  + setModel(File file): void
  + resetView(): void
```

```
<<Interface>>
ControllerFeatures
  + addComputerPlayer(String playerName, String playerLocation, int itemLimit): void
  + addPlayer(String playerName, String playerLocation, int itemLimit): void
  + attack(Strin itemName): void
  + describePlayer(): void
  + lookAround(): void
  + getTurnInformation(): String
  + move(String nameOfSpace): void
  + movePet(String nameOfSpace): void
  + pickUp(String itemName): void
  + hitDetection(Double x, Double y): SpaceViewModel
  + setModel(File file): void
  + getCurrentPlayer(): PlayerViewModel
  + startGame(int turnCount): void
  + spawningRooms(): List<SpaceViewModel>
  + restartGame(): void
  + gameStarted(): boolean
```

Facade interface

```
<<Interface>>
World
  + printWorldImageToDisk(): String
  + worldImage(): BufferedImage
  + hitDetection(Double x, Double y): Space
  + moveTarget(): void
  + nextTurn(): void
  + getSpaces(): List<Space>
  + getTheSpaceByName(string nameOfSpace): Space
  + getNeighborsOfaSpace(int indexOfSpace): List<Space>
  + addPlayer(String playerName, Space player location, int itemLimit): void
  + addComputerPlayer(String playerName, Space player location, int itemLimit): void
  + getMaxNumberOfTurns(): int
  + setMaxNumberOfTurns(int maxTurns): void
  + getAvailableLocations(): String
  + getCurrentPlayerTurnInfo(): String
  + getCurrentPlayer(): Player
  + getTurnTotal(): int
  + isGameOver(): boolean
  + getPlayerCount(): int
```

```
<<Interface>>
GamePlayCommand
  + execute(World game): String
```

```
AddPlayer
  - playerName: String
  - playerLocation: String
  - itemLimt: String
  + AddPlayer(String playerName, String playerLocation, int itemLimit): void
  + execute(World game): String
```

```
SaveWorldImage
  + execute(World game): String
```

```
AddComputerPlayer
  - playerName: String
  - playerLocation: String
  - itemLimt: String
  + AddPlayer(String playerName, String playerLocation, int itemLimit): void
  + execute(World game): String
```

```
MovePet
  - nameOfSpace: String
  + movePet(String nameOfSpace): void
  + execute(World game): String
```

```
DescribeSpace
  - nameOfSpace: String
  + DescribeSpace(String spaceName): void
  + execute(World game): String
```

```
PickUpItem
  -memberName
  - itemName: String
  + PickUpAnItem(String name): void
  + execute(World game): String
```

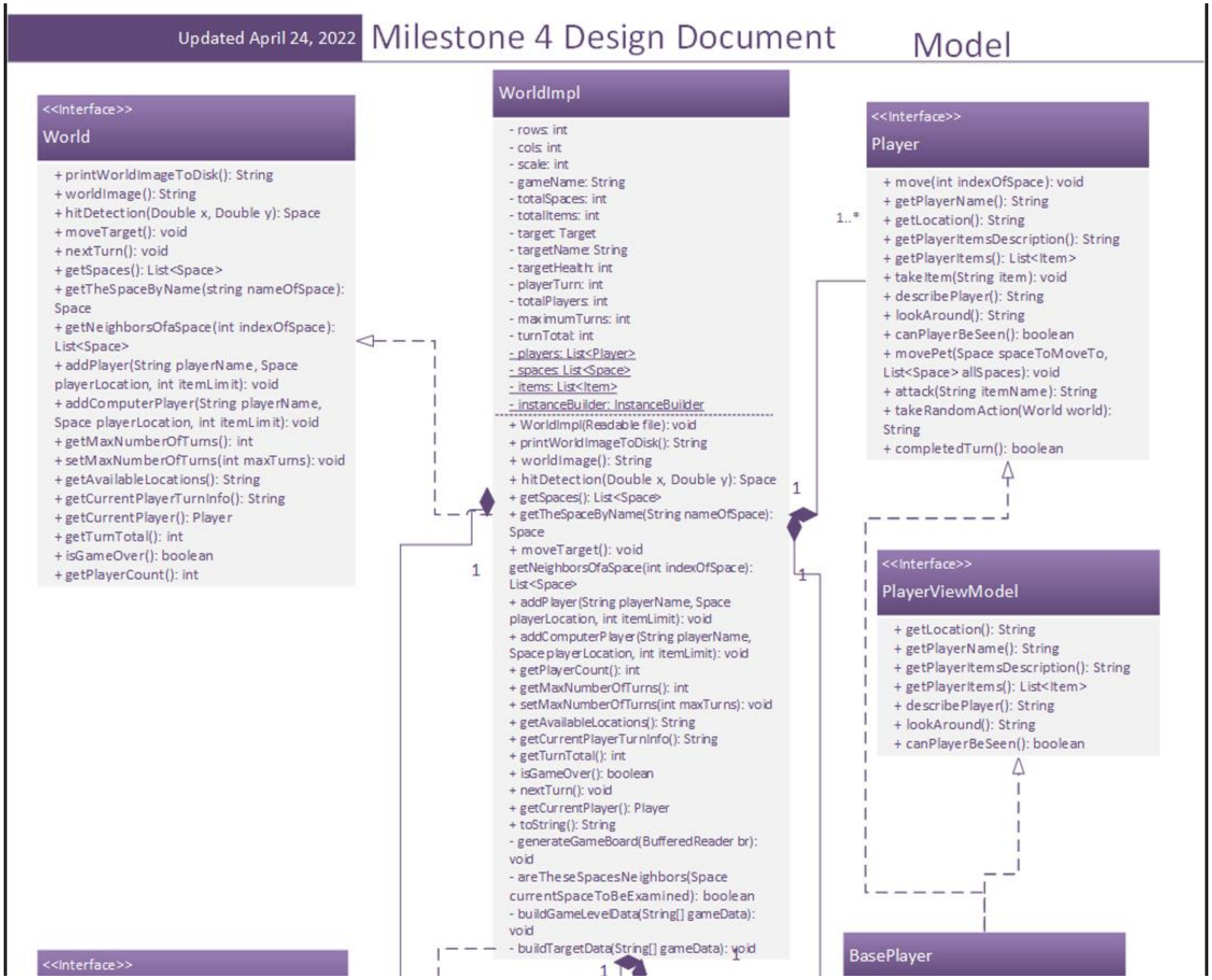
```
turnInformation
  + execute(World game): String
```

```
DescribePlayer
  + execute(World game): String
```

```
LookAround
  + execute(World game): String
```

```
Move
  - nameOfSpace: String
  + move(String nameOfSpace): void
  + execute(World game): String
```

```
Attack
  -itemName: String
  + Attack(String itemName): void
  + execute(World game): String
```



Updated April 24, 2022 Milestone 4 Design Document

Model

```
<<Interface>>
World
+ printWorldImageToDisk(): String
+ worldImage(): String
+ hitDetection(Double x, Double y): Space
+ moveTarget(): void
+ nextTurn(): void
+ getSpaces(): List<Space>
+ getTheSpaceByName(string nameOfSpace):
Space
+ getNeighborsOfaSpace(int indexOfSpace):
List<Space>
+ addPlayer(String playerName, Space
playerLocation, int itemLimit): void
+ addComputerPlayer(String playerName,
Space playerLocation, int itemLimit): void
+ getMaxNumberOfTurns(): int
+ setMaxNumberOfTurns(int maxTurns): void
+ getAvailableLocations(): String
+ getCurrentPlayerTurnInfo(): String
+ getCurrentPlayer(): Player
+ getTurnTotal(): int
+ isGameOver(): boolean
+ getPlayerCount(): int
```

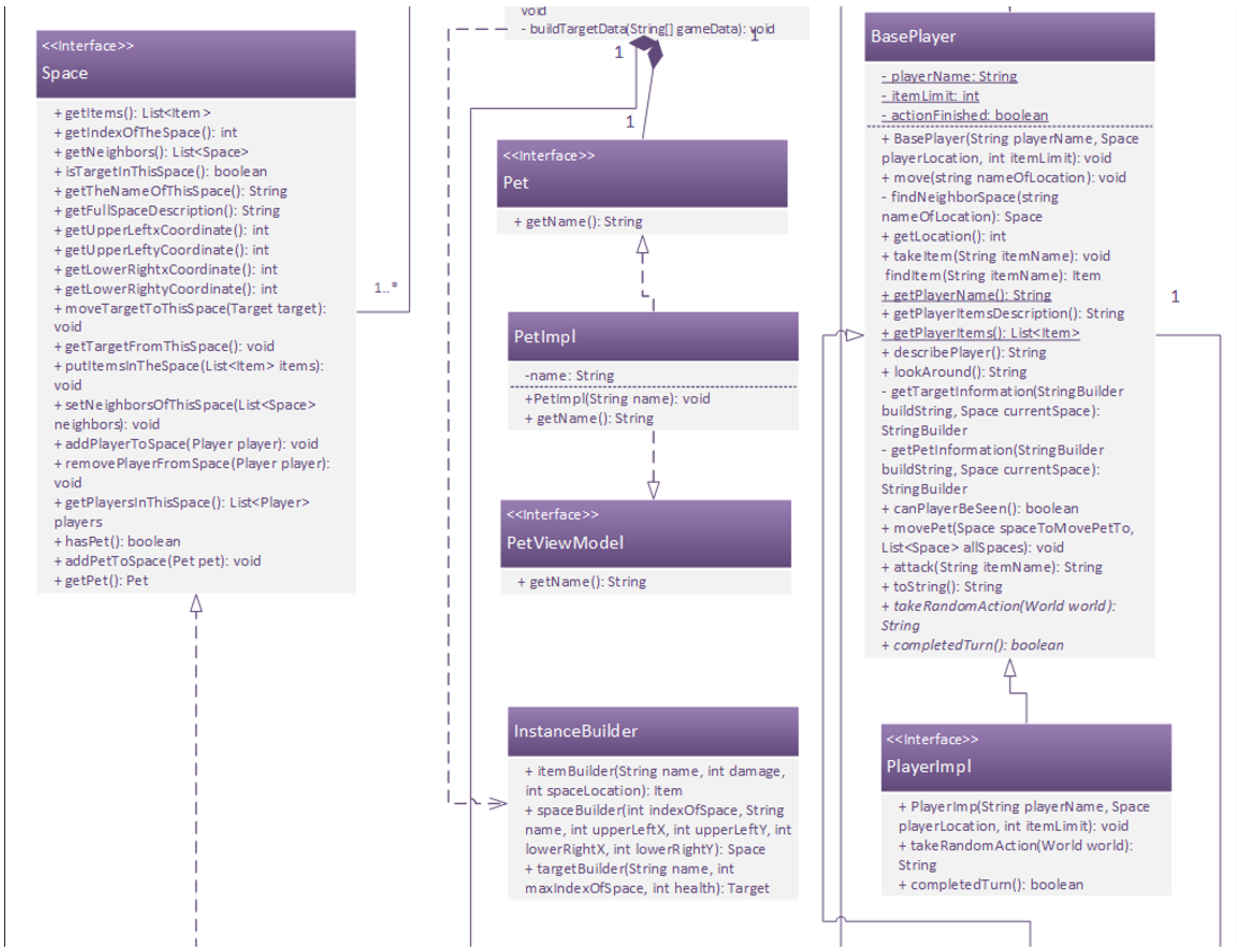
```
WorldImpl
- rows: int
- cols: int
- scale: int
- gameName: String
- totalSpaces: int
- totalItems: int
- target: Target
- targetName: String
- targetHealth: int
- playerTurn: int
- totalPlayers: int
- maximumTurns: int
- turnTotal: int
- players: List<Player>
- spaces: List<Space>
- items: List<Item>
- instanceBuilder: InstanceBuilder
+ WorldImpl(Readable file): void
+ printWorldImageToDisk(): String
+ worldImage(): String
+ hitDetection(Double x, Double y): Space
+ getSpaces(): List<Space>
+ getTheSpaceByName(String nameOfSpace):
Space
+ moveTarget(): void
getNeighborsOfaSpace(int indexOfSpace):
List<Space>
+ addPlayer(String playerName, Space
playerLocation, int itemLimit): void
+ addComputerPlayer(String playerName,
Space playerLocation, int itemLimit): void
+ getPlayerCount(): int
+ getMaxNumberOfTurns(): int
+ setMaxNumberOfTurns(int maxTurns): void
+ getAvailableLocations(): String
+ getCurrentPlayerTurnInfo(): String
+ getTurnTotal(): int
+ isGameOver(): boolean
+ nextTurn(): void
+ getCurrentPlayer(): Player
+ toString(): String
- generateGameBoard(BufferedReader br):
void
- areTheseSpacesNeighbors(Space
currentSpaceToBeExamined): boolean
- buildGameLevelData(String[] gameData):
void
- buildTargetData(String[] gameData): void
```

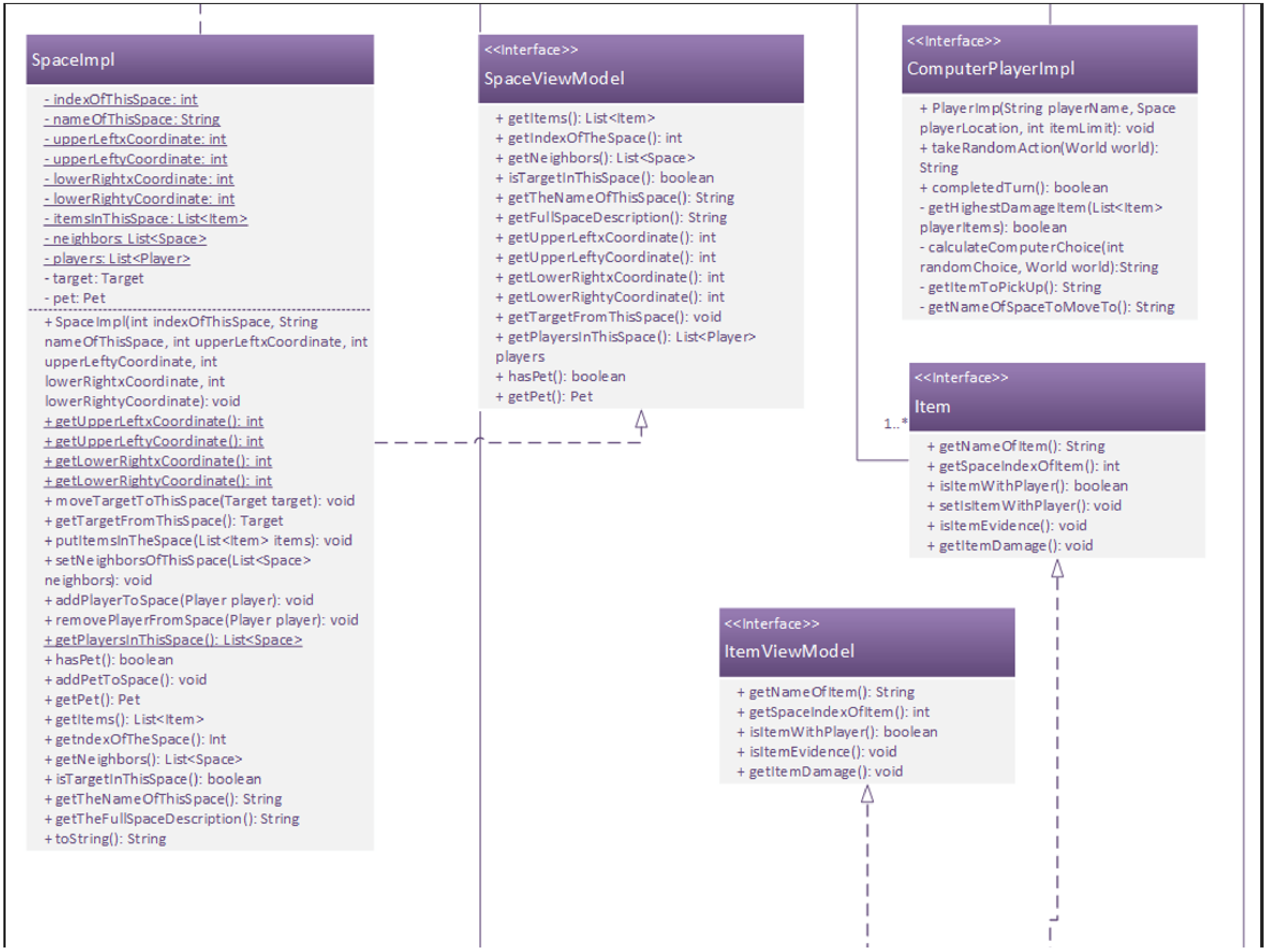
```
<<Interface>>
Player
+ move(int indexOfSpace): void
+ getPlayerName(): String
+ getLocation(): String
+ getPlayerItemsDescription(): String
+ getPlayerItems(): List<Item>
+ takeItem(String item): void
+ describePlayer(): String
+ lookAround(): String
+ canPlayerBeSeen(): boolean
+ movePet(Space spaceToMoveTo,
List<Space> allSpaces): void
+ attack(String itemName): String
+ takeRandomAction(World world):
String
+ completedTurn(): boolean
```

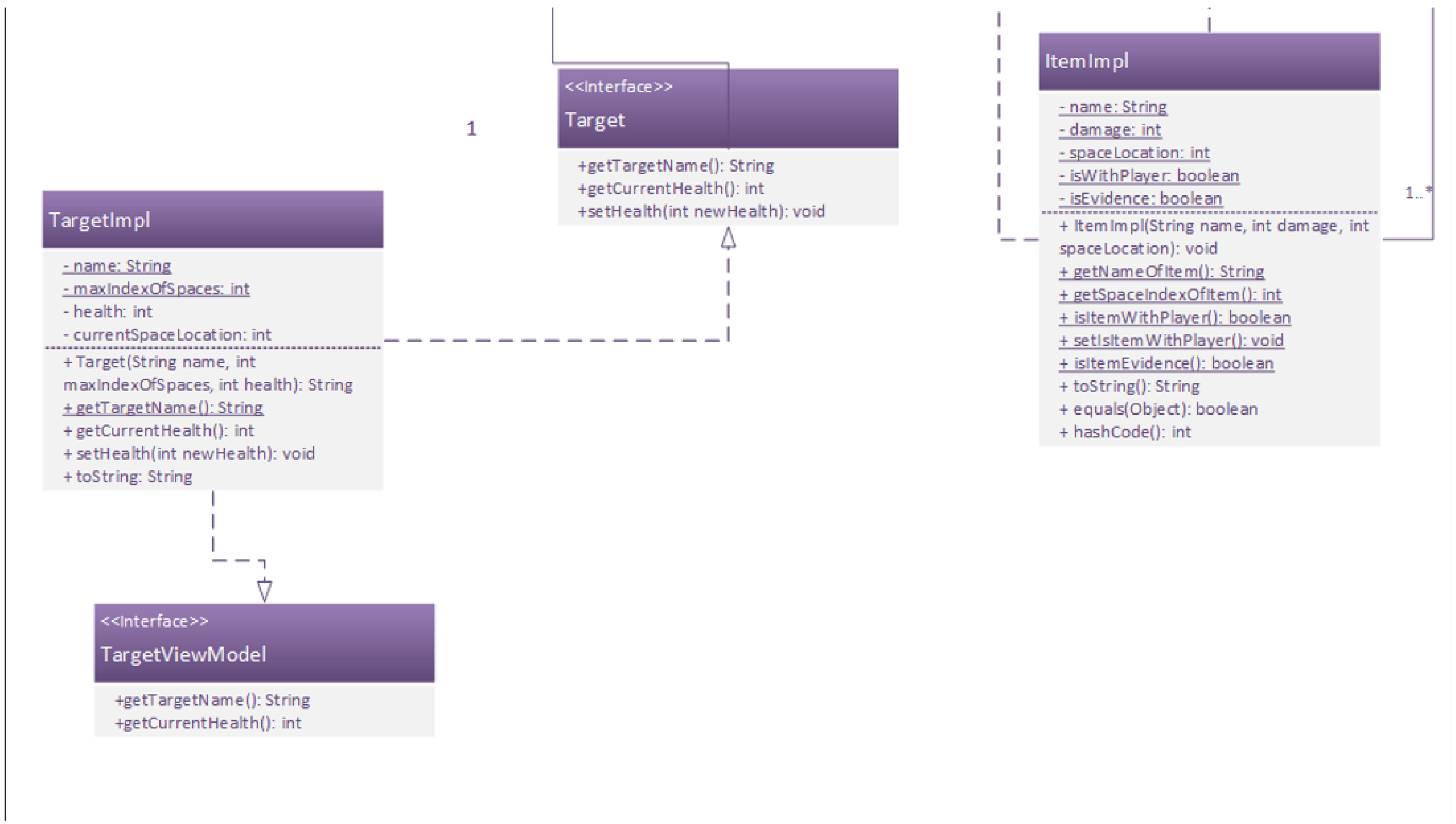
```
<<Interface>>
PlayerViewModel
+ getLocation(): String
+ getPlayerName(): String
+ getPlayerItemsDescription(): String
+ getPlayerItems(): List<Item>
+ describePlayer(): String
+ lookAround(): String
+ canPlayerBeSeen(): boolean
```

```
BasePlayer
```

```
<<Interface>>
```


void
- buildTargetData(String[] gameData): void
1
1
1
1..*
1
<<Interface>>
Space
+ getItems(): List<Item>
+ getIndexOfTheSpace(): int
+ getNeighbors(): List<Space>
+ isTargetInThisSpace(): boolean
+ getTheNameOfThisSpace(): String
+ getFullSpaceDescription(): String
+ getUpperLeftxCoordinate(): int
+ getUpperLeftyCoordinate(): int
+ getLowerRightxCoordinate(): int
+ getLowerRightyCoordinate(): int
+ moveTargetToThisSpace(Target target): void
+ getTargetFromThisSpace(): void
+ putItemsInTheSpace(List<Item> items): void
+ setNeighborsOfThisSpace(List<Space> neighbors): void
+ addPlayerToSpace(Player player): void
+ removePlayerFromSpace(Player player): void
+ getPlayersInThisSpace(): List<Player> players
+ hasPet(): boolean
+ addPetToSpace(Pet pet): void
+ getPet(): Pet
<<Interface>>
Pet
+ getName(): String
PetImpl
-name: String
+PetImpl(String name): void
+ getName(): String
<<Interface>>
PetViewModel
+ getName(): String
InstanceBuilder
+ itemBuilder(String name, int damage, int spaceLocation): Item
+ spaceBuilder(int indexOfSpace, String name, int upperLeftX, int upperLeftY, int lowerRightX, int lowerRightY): Space
+ targetBuilder(String name, int maxIndexOfSpace, int health): Target
BasePlayer
- playerName: String
- itemLimit: int
- actionFinished: boolean
+ BasePlayer(String playerName, Space playerLocation, int itemLimit): void
+ move(string nameOfLocation): void
- findNeighborSpace(string nameOfLocation): Space
+ getLocation(): int
+ takeItem(String itemName): void
findItem(String itemName): Item
+ getPlayerName(): String
+ getPlayerItemsDescription(): String
+ getPlayerItems(): List<Item>
+ describePlayer(): String
+ lookAround(): String
- getTargetInformation(StringBuilder buildString, Space currentSpace): StringBuilder
- getPetInformation(StringBuilder buildString, Space currentSpace): StringBuilder
+ canPlayerBeSeen(): boolean
+ movePet(Space spaceToMovePetTo, List<Space> allSpaces): void
+ attack(String itemName): String
+ toString(): String
+ takeRandomAction(World world): String
+ completedTurn(): boolean
<<Interface>>
PlayerImpl
+ PlayerImp(String playerName, Space playerLocation, int itemLimit): void
+ takeRandomAction(World world): String
+ completedTurn(): boolean

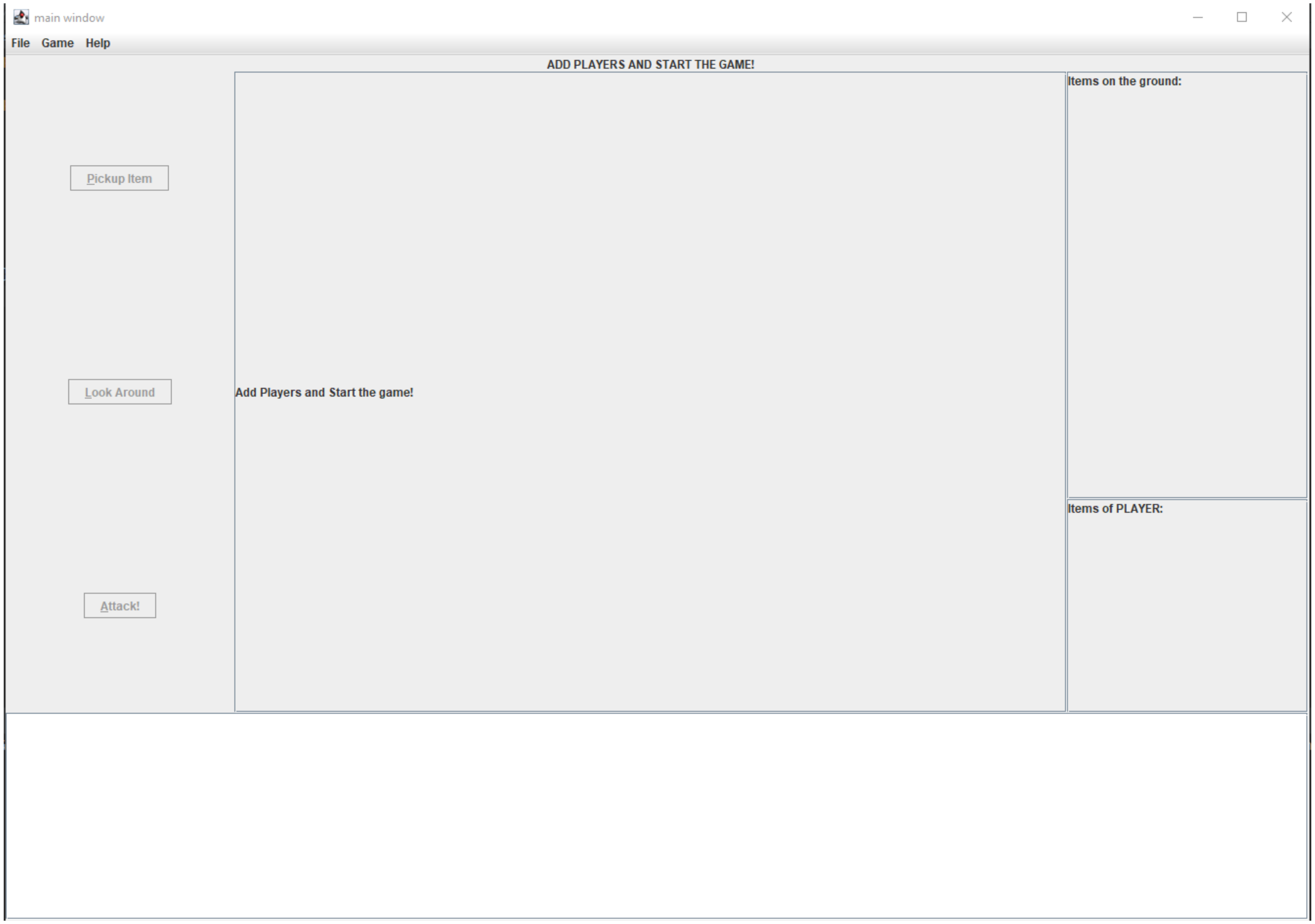

SpaceImpl
- indexOfThisSpace: int
- nameOfThisSpace: String
- upperLeftxCoordinate: int
- upperLeftyCoordinate: int
- lowerRightxCoordinate: int
- lowerRightyCoordinate: int
- itemsInThisSpace: List<Item>
- neighbors: List<Space>
- players: List<Player>
- target: Target
- pet: Pet
+ SpaceImpl(int indexOfThisSpace, String nameOfThisSpace, int upperLeftxCoordinate, int upperLeftyCoordinate, int lowerRightxCoordinate, int lowerRightyCoordinate): void
+ getUpperLeftxCoordinate(): int
+ getUpperLeftyCoordinate(): int
+ getLowerRightxCoordinate(): int
+ getLowerRightyCoordinate(): int
+ moveTargetToThisSpace(Target target): void
+ getTargetFromThisSpace(): Target
+ putItemsInTheSpace(List<Item> items): void
+ setNeighborsOfThisSpace(List<Space> neighbors): void
+ addPlayerToSpace(Player player): void
+ removePlayerFromSpace(Player player): void
+ getPlayersInThisSpace(): List<Space>
+ hasPet(): boolean
+ addPetToSpace(): void
+ getPet(): Pet
+ getItems(): List<Item>
+ getndexOfTheSpace(): Int
+ getNeighbors(): List<Space>
+ isTargetInThisSpace(): boolean
+ getTheNameOfThisSpace(): String
+ getTheFullSpaceDescription(): String
+ toString(): String
<<Interface>>
SpaceViewModel
+ getItems(): List<Item>
+ getIndexOfTheSpace(): int
+ getNeighbors(): List<Space>
+ isTargetInThisSpace(): boolean
+ getTheNameOfThisSpace(): String
+ getFullSpaceDescription(): String
+ getUpperLeftxCoordinate(): int
+ getUpperLeftyCoordinate(): int
+ getLowerRightxCoordinate(): int
+ getLowerRightyCoordinate(): int
+ getTargetFromThisSpace(): void
+ getPlayersInThisSpace(): List<Player> players
+ hasPet(): boolean
+ getPet(): Pet
<<Interface>>
ComputerPlayerImpl
+ PlayerImp(String playerName, Space playerLocation, int itemLimit): void
+ takeRandomAction(World world): String
+ completedTurn(): boolean
- getHighestDamageItem(List<Item> playerItems): boolean
- calculateComputerChoice(int randomChoice, World world):String
- getItemToPickUp(): String
- getNameOfSpaceToMoveTo(): String
<<Interface>>
Item
1..*
+ getNameOfItem(): String
+ getSpaceIndexOfItem(): int
+ isItemWithPlayer(): boolean
+ setIsItemWithPlayer(): void
+ isItemEvidence(): void
+ getItemDamage(): void
<<Interface>>
ItemViewModel
+ getNameOfItem(): String
+ getSpaceIndexOfItem(): int
+ isItemWithPlayer(): boolean
+ isItemEvidence(): void
+ getItemDamage(): void


1
<<Interface>>
Target
+getTargetName(): String
+getCurrentHealth(): int
+setHealth(int newHealth): void
ItemImpl
- name: String
- damage: int
- spaceLocation: int
- isWithPlayer: boolean
- isEvidence: boolean
+ ItemImpl(String name, int damage, int spaceLocation): void
+ getNameOfItem(): String
+ getSpaceIndexOfItem(): int
+ isItemWithPlayer(): boolean
+ setIsItemWithPlayer(): void
+ isItemEvidence(): boolean
+ toString(): String
+ equals(Object): boolean
+ hashCode(): int
1..*
TargetImpl
- name: String
- maxIndexOfSpaces: int
- health: int
- currentSpaceLocation: int
+ Target(String name, int maxIndexOfSpaces, int health): String
+ getTargetName(): String
+ getCurrentHealth(): int
+ setHealth(int newHealth): void
+ toString: String
<<Interface>>
TargetViewModel
+getTargetName(): String
+getCurrentHealth(): int

View Design

Instead of the text-based controller, a new graphical controller is added to replace the one found in milestone 3. This new controller strictly follows a traditional MVC pattern, which facilitates the communication between the view and the model. More specifically, all user actions in the view will be forwarded into the controller (which itself only exposes a features interface).

Opening Screen



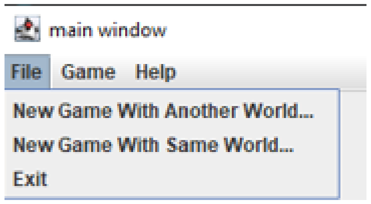
ADD PLAYERS AND START THE GAME!

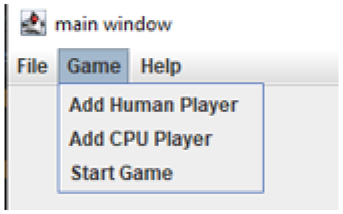
Add Players and Start the game!

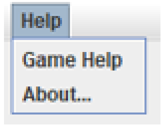
Items on the ground:

Items of PLAYER:

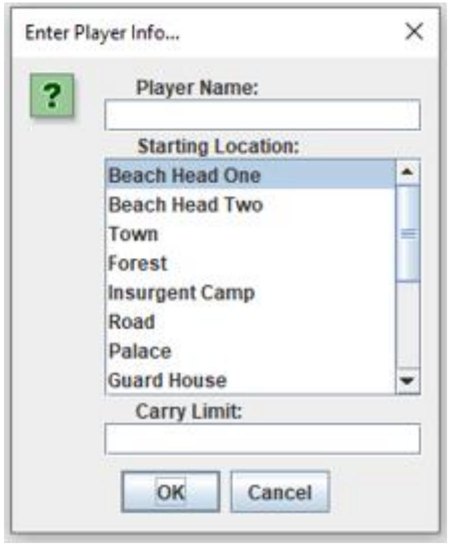
Menus



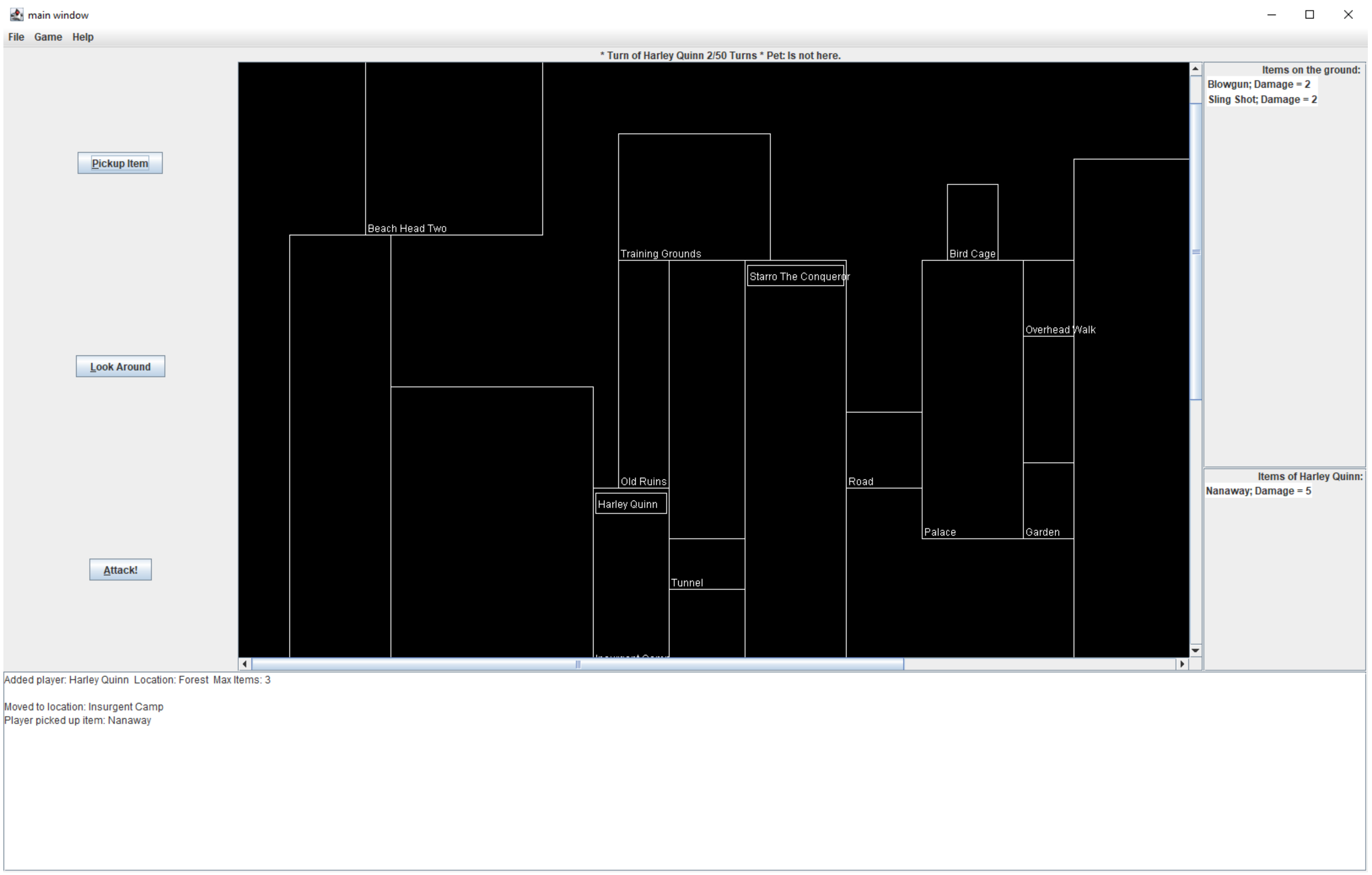




Add Player/Computer Player Dialog



In Game Play



* Turn of Harley Quinn 2/50 Turns * Pet: Is not here.

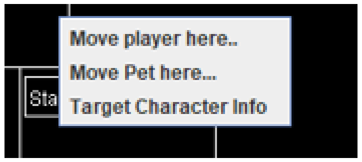
Items on the ground:
Blowgun; Damage = 2
Sling Shot; Damage = 2

Items of Harley Quinn:
Nanaway; Damage = 5

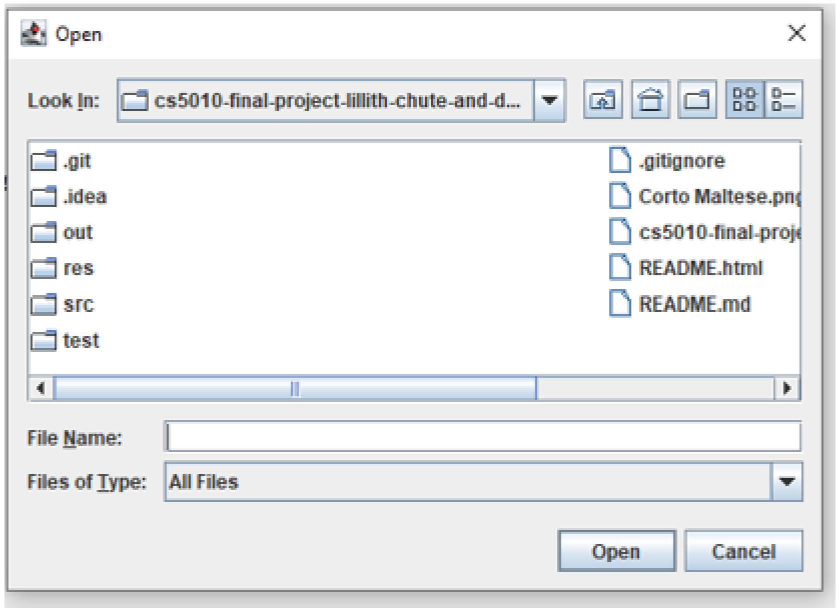
```
Added player: Harley Quinn  Location: Forest  Max Items: 3

Moved to location: Insurgent Camp
Player picked up item: Nanaway
```

Right Click Context Menu During Play



Open File Dialog



Model Design

In this milestone we will be using Lillith's model as the basis for the game. There is very little game logic that was maintained in the controller as a big part of the exercise for milestone 3 was to separate the game logic as much as possible from the controller and maintain that in the model. The controller just queried the model to see if the game was over to determine whether or not to keep playing. It queried the model to figure out if the player whose turn it is was human or not and queried the model to figure out whose turn it is. Therefore, the controller doesn't manage the game, it just told the model what to do when a user entered an action in the console.

Some model methods will be refactored, such as WorldImpl.printWorldImageToDisk(). Instead of performing I/O operations or internal state mutations, these methods will now return raw output back to the controller. The controller will process these data and push the corresponding changes to update the view.

In order to further decouple the model, the controller and the view, all model classes will now have a corresponding ViewModel interface (such as ItemImpl and ItemViewModel). These new interfaces allow read operations but restricts write operations, so that no data can be mutated when obtained elsewhere in a view or a controller.

There was a fair bit of refactoring that was required that was not foreseen during the initial design phase. A great deal of this was the result of not being familiar at all with Java Swing components and what they needed from the controller which had a ripple effect down into the model and how it needed to interact with the controller and the data required. Some of the changes are as follows:

1. Redesigned the way that the Player is represented:
    a. The Player and computer player was distinguished through the use of a boolean flag in Milestone 3.
    b. The player has to be redesigned to by polymorphic
        i. The player has an interface
        ii. The BasePlayer class holds all the common actions

    iii. The human player extends the BasePlayer class and actions specific to humans resides there

    iv. The computer player class has the random functionality implemented there.

2. Significant modifications were made to the World class as a lot of the computer player management was done in that class and all of it was moved to the Computer Player class.
3. Updated functionality in the Command Pattern to better interact with the View
4. Made significant code changes to make the rendering of the image for the game board look since and be properly rendered and aligned.
5. Also made a lot of changes in that same piece of code to place the player name plates inside the rooms they are located in.
6. Created ViewModels interfaces for all the models to provide read-only access to the controller to create further separation of concerns and safety
7. Mock model classes and a Mock view class is added to the project. They are capable of logging method calls and are used in controller testing.

In addition to the above was the implementation of the View. There were a number of additional design changes from the initial design to this. The above list is not all inclusive but represents the most significant work.

Additionally, there will be other changes related to fixing issues from the manual grading and refining the graphical representation of the world.

Test Plan

My understanding is that the command pattern objects will be tested indirectly through the controller when the controller takes in commands from the user and those commands are forwarded by the command object to the model. The bulk of the new testing will be for the use of the GraphicalController as the View doesn't need to be tested.

MILESTONE 4 Controller Tests:

Testing design for GraphicalController:

Testing GraphicalController with mock model and mock view	Input	Expected
addComputerPlayer (invalid)	null playerName, valid playerLocation Valid Item limit	IllegalArgumentException

Testing GraphicalController with mock model and mock view	Input	Expected
addComputerPlayer (invalid)	valid playerName, invalid playerLocation Valid Item limit	IllegalArgumentException
addComputerPlayer (valid)	valid playerName, valid playerLocation Valid Item limit	“addComputerPlayer called, playerName = Computer Player, "
addPlayer (invalid)	null playerName, valid playerLocation Valid Item limit	IllegalArgumentException
addPlayer (invalid)	valid playerName, invalid playerLocation Valid Item limit	IllegalArgumentException

Testing GraphicalController with mock model and mock view	Input	Expected
addPlayer (invalid)	valid playerName, valid playerLocation Valid Item limit	“addComputerPlayer called, playerName = Computer Player, "
attack	Valid item	"attack called, itemName=Poke in the Eye"addComputerPlayer called, playerName = Computer Player, "
describePlayer	N/A	"describePlayer called."
lookaround	N/A	"lookAround called."
getTurnInformation()	null	"getCurrentPlayerTurnInfo called."
move (invalid)	null	IllegalArgumentException
move (valid)	Null	"move called, nameOfSpace=Forest"
movePet (invalid)	null	IllegalArgumentException
movePet (valid)	valid nameOfSpace	"movePet called, spaceToMoveTo"

Testing GraphicalController with mock model and mock view	Input	Expected
pickup (invalid)	null	IllegalArgumentException
pickup (valid)	valid itemName	"takeItem called."
hitDetection(Double x, Double y)	Valid coords	"hitDetection called, x=23.000000, y=46.000000"
setView (invalid)	null	IllegalArgumentException
setView (valid)	valid MainForm object	"welcome called."
setModel (invalid)	null	IllegalArgumentException
setModel (valid)	valid MainForm object	"welcome called."
restartGame()	null	"promptError called. msg = No World is loaded!"
resetView()	null	"setStartedState called. started = false"
gameStarted()	null	"welcome called."

Testing GraphicalController with mock model and mock view	Input	Expected
startGame(valid)	100	"promptError called. msg = Not Enough players to start the game!"
getCurrentPlayer()	null	"getCurrentPlayer called."
spawningRooms()	Null	"getSpaces called."

Testing design for PetImpl:

Testing: Constructor	Input	Expected
Invalid constructor null parameters	PetImpl(null, null)	IllegalArgumentException
Invalid name	PetImpl(null, 3)	IllegalArgumentException
Invalid location	PetImpl(“Sparky”, -3)	IllegalArgumentException
Valid input	PetImpl(“Sparky”, 3)	

Testing: move ()	Input	Expected
Invalid room name	movePet(“Fire pit”)	IllegalArgumentException
Invalid null room	movePet(null)	IllegalArgumentException
Invalid empty string	movePet(“”)	IllegalArgumentException
Invalid same room	movePet(“Jotunheim”)	IllegalArgumentException
Valid move	movePet(“Guard house”)	Returns successful move to Guard House

Testing: wander()	Input	Expected
Moves to next room in DFS		Next expected room

Testing design for ComputerPlayerImpl:

Testing: Constructor	Input	Expected
Invalid constructor null name	PlayerImpl(null)	IllegalArgumentException
Invalid constructor empty name	PlayerImpl (“”)	IllegalArgumentException
Valid constructor	PlayerImpl (“Harley”)	“Harley”

Testing: calculateComputerMove()	Input	Expected
Decides to move	calculateComputerMove()	Computer player moves to neighboring room
Looks around	calculateComputerMove()	Computer decides to look around
Describes player	calculateComputerMove()	Computer calls describePlayer
Picks up an item	calculateComputerMove()	Picks up an item

Testing design for PlayerImpl:

Testing: Constructor	Input	Expected
Invalid constructor null name	PlayerImpl(null)	IllegalArgumentException
Invalid constructor empty name	PlayerImpl (“”)	IllegalArgumentException
Valid constructor	PlayerImpl (“Harley”)	“Harley”

Testing: move()	Input	Expected

Invalid room name	move(“Fire pit”)	IllegalArgumentException
Invalid null room	move(null)	IllegalArgumentException
Invalid empty string	move(“”)	IllegalArgumentException
Invalid same room	move(“Jotunheim”)	IllegalArgumentException
Invalid room non neighbor	move(“Forrest”)	IllegalArgumentException
Valid move to neighboring space	move(“Guard house”)	Call describePlayer(); expect room name to be “Guard house”

Testing: describePlayer()	Input	Expected
Describe the player	describePlayer()	The player name, the space the player is in, the items the player is carrying, the total amount of items the player can carry.

Testing: lookAround()	Input	Expected
Player looks around	lookAround()	The space the player is in, the items in the space, any neighboring spaces, and the target if in the space or neighbors.

Testing: takeItem()	Input	Expected
Invalid item name	takeItem(“Sponge”)	IllegalArgumentException
Invalid null item	takeItem (null)	IllegalArgumentException
Invalid empty string	takeItem (“”)	IllegalArgumentException
Item limit exceeded	takeItem (“Pistol”)	IllegalArgumentException
Item exists but isn’t in this room	takeItem (“Javelin”)	IllegalArgumentException
Valid item	takeItem (“Polka-dot”)	Call describePlayer(); expect item name to be “Polka-dot”

Testing: hashcode()	Input	Expected
equals	playerOne.hashCode(), playerSame.hashCode()	True
Not equals	playerOne.hashCode() playerTwo.hashCode()	False

Testing: equals()	Input	Expected
equals	playerOne.equals(playerSame)	True
Not equals	playerOne.equals(playerTwo)	False
Testing: toString()	**Input**	**Expected**
String representation of constructor	PlayerImpl("Corto Maltese")	"Corto Maltese"

Testing design for ManageGameImpl:

Testing: Constructor	Input	Expected
Invalid constructor null filepath	ManageGameImpl(null)	IllegalArgumentException
Invalid constructor empty filepath	ManageGameImpl("")	IllegalArgumentException
A space is outside of the expected grid	ManageGameImpl(filepath)	IllegalArgumentException
A space has negative coordinates	ManageGameImpl(filepath)	IllegalArgumentException
Valid constructor	ManageGameImpl(filepath)	"40;40;Corto Maltese"

Testing: DisplayBoard()	Input	Expected

Test that an image is produced	game.displayBoard()	Assert that the image file exists Assert that the mime type is image

Testing: moveTarget()	Input	Expected
Tests if the manage can move the target	game.moveTarget() game.moveTarget()	Assert target starts at index 0 Assert after moving twice it is at index 2

Testing: getSpaces()	Input	Expected
Get the number of spaces in the file	game.getSpaces()	Assert there are 9 spaces Assert the ninth space is Jotunheim

Testing: targetLocation()	Input	Expected
Get current location of the target. Move target 9 times and check again	game.targetLocation()	Assert target is at index 0 Assert target is at index 8

Testing: ToString	Input	Expected
Get string representation of rows, columns, world name	game.toString()	“40;40;Corto Maltese”

Testing: getDescriptionOfSpace()	Input	Expected
Give valid name of space	getDescriptionOfSpace (“Jotunheim”)	Items: C4, Bloodsport, Polka-Dots, Rats Neighbors: Guard house
Incorrect space name	getDescriptionOfSpace (“Valhalla”)	IllegalArgumentException
Null	getDescriptionOfSpace (null)	IllegalArgumentException

Incorrect index	getDescriptionOfSpace (23)	IllegalArgumentException
Negative index	getDescriptionOfSpace (-1)	IllegalArgumentException
Valid location	getDescriptionOfSpace (8)	Items: C4, Bloodsport, Polka-Dots, Rats Neighbors: Guard house

Testing: getNeighborsOfaSpace()	Input	Expected
Give valid index of a space	getNeighborsOfaSpace (8) getNeighborsOfaSpace (5)	Assert 1 and Assert 2
Bad index	getNeighborsOfaSpace (23)	IllegalArgumentException
Negative index	getNeighborsOfaSpace (-1)	IllegalArgumentException
Valid name	getNeighborsOfaSpace (“Jotunheim”) getNeighborsOfaSpace (“Road”)	Assert 1 and assert 2
Invalid name	getNeighborsOfaSpace (“Valhalla”)	IllegalArgumentException
Null	getNeighborsOfaSpace (null)	IllegalArgumentException

Testing: getNameOfSpaceByName ()	Input	Expected
Should get valid name back	getNameOfSpaceByIndex(8)	“Jotunheim”
Index is negative	getNameOfSpaceByIndex(-1)	IllegalArgumentException
Index is out of bounds	getNameOfSpaceByIndex(21)	IllegalArgumentException

Testing: move()	Input	Expected
Invalid room name	move(“Fire pit”)	IllegalArgumentException
Invalid null room	move(null)	IllegalArgumentException
Invalid empty string	move(“”)	IllegalArgumentException
Invalid same room	move(“Jotunheim”)	IllegalArgumentException

Invalid room non neighbor	move("Forrest")	IllegalArgumentException
Valid move to neighboring space	move("Guard house")	Call describePlayer(); expect room name to be "Guard house"

Testing: describePlayer()	Input	Expected
Describe the player	describePlayer()	The player name, the space the player is in, the items the player is carrying, the total amount of items the player can carry.

Testing: lookAround()	Input	Expected
Player looks around	lookAround()	The space the player is in, the items in the space, any neighboring spaces, and the target if in the space or neighbors.

Testing: takeItem()	Input	Expected
Invalid item name	takeItem("Sponge")	IllegalArgumentException
Invalid null item	takeItem (null)	IllegalArgumentException
Invalid empty string	takeItem ("")	IllegalArgumentException
Item limit exceeded	takeItem ("Pistol")	IllegalArgumentException
Item exists but isn't in this room	takeItem ("Javelin")	IllegalArgumentException
Valid item	takeItem ("Polka-dot")	Call describePlayer(); expect item name to be "Polka-dot"

Testing: startOfTurnInfo()	Input	Expected

Gets information about the player and the room they are in		Returns immediate room information and player detail.

Testing: canPlayerBeSeen()	Input	Expected
Player can be seen by another player		True
Player can be seen by multiple players		True
Pet is in the room but players are in neighboring room		False
No players in neighboring rooms and no pet in room		False
Pet is in the room and no players in neighboring rooms		False

Testing: attack()	Input	Expected
Attack with no target in the room	attack("Knife")	IllegalStateException
Attack with item you don't have	attack("Sloppy Joe")	IllegalArgumentException
Attack but can be seen	attack("Knife")	IllegalStateException
Attack successful but doesn't kill target	attack("Knife")	Attack hits but doesn't kill target, target hit points remaining displayed
Attack kills target	attack("Knife")	Game ends

Testing: movePet()	Input	Expected
Invalid room name	movePet("Fire pit")	IllegalArgumentException
Invalid null room	movePet(null)	IllegalArgumentException
Invalid empty string	movePet("")	IllegalArgumentException
Invalid same room	movePet("Jotunheim")	IllegalArgumentException
Valid move	movePet("Guard house")	Returns successful move to Guard House

Testing design for SpaceImpl:

Testing: Constructor	Input	Expected
Valid input	spaceBuilder(2, “Jotunheim”, 1, 1, 4, 4)	“2;Jotunheim;1;1;4;4”
upperLeftX <= 0	spaceBuilder(2, “Jotunheim”, -1, 1, 4, 4)	IllegalArgumentException
upperLeftY <= 0	spaceBuilder(2, “Jotunheim”, 1, -1, 4, 4)	IllegalArgumentException
lowerRightX <= 0	spaceBuilder(2, “Jotunheim”, 1, 1, -4, 4)	IllegalArgumentException
lowerRightY <= 0	spaceBuilder(2, “Jotunheim”, 1, 1, 4, -4)	IllegalArgumentException
LowerRightY < 0 and upperLeftX < 0	spaceBuilder(2, “Jotunheim”, -1, 1, 4, -4)	IllegalArgumentException
LowerRightX < 0 and upperLeftX < 0	spaceBuilder(2, “Jotunheim”, -1, 1, -4, 4)	IllegalArgumentException
Negative coordinates	spaceBuilder(2, “Jotunheim”, -1, -1, -4, -4)	IllegalArgumentException
Negative space index	spaceBuilder(-2, “Jotunheim”, 1, 1, 4, 4)	IllegalArgumentException
Name is blank	spaceBuilder(2, “”, 1, 1, 4, 4)	IllegalArgumentException
Name is null	spaceBuilder(2, null, 1, 1, 4, 4)	IllegalArgumentException
upperLeftX > lower right Y	spaceBuilder(2, “Jotunheim”, 100, 1, 40, 4)	IllegalArgumentException
upperLeftY > lowerRightY	spaceBuilder(2, “Jotunheim”, 100, 100, 4, 4)	IllegalArgumentException
UpperLeftx > lowerRightX and upperLeftY > lowerRightY	spaceBuilder(2, “Jotunheim”, 1, 10, 4, 4)	IllegalArgumentException

Testing: getItems()	Input	Expected
Get the number of items in this space	getItems(items)	Assert number 2
Get the names of the items	getItems(items)	“Baseball Bat;2;2;Machete;3;2;”
Create new empty list of items space now has no items	getItems(items)	0

Testing: getIndexOfTheSpace()	Input	Expected
Find the index of this space	getIndexOfTheSpace()	2

Testing: isTargetInThisSpace()	Input	Expected
Check space where the target is not	isTargetInThisSpace(target)	False
Check a space where the target isn’t and then move the target so it is	isTargetInThisSpace(target) moveTarget() x 2 isTargetInThisSpace(target)	False and then True

Testing: getTheNameOfThisSpace()	Input	Expected
Get name of this current space	getTheNameOfThisSpace()	“Jotunheim”

Testing: getNeighbors()	Input	Expected
Get the neighbors of dining hall	diningHall.getNeighbors(spaces)	"Tennesse Room;Parlor;Kitchen;Wine Cellar;Drawing Room;Armory;Billiard Room;Trophy Room;"
Get neighbors of the kitchen	kitchen.getNeighbors(spaces)	"Dining Hall;Parlor;Wine Cellar;"
Get neighbors of the Armory	armory.getNeighbors(spaces)	"Dining Hall;Drawing Room;Billiard Room;"

Testing: getTheFullSpaceDescription()	Input	Expected
Get the full description of this space which includes name of this space, items, and neighbors	armory.getTheFullSpaceDescription(itesm, spaces)	The return value should include Armory, polka-dots, detachable arms, dining hall, drawing room, and billiard room

Testing: Hashcode	Input	Expected
equals	Dininghall.hashCode, diningHallSame.hashCode	True
Not equals	diningHall.hashCode() kitchen.hashCode()	False

Testing: Equals()	Input	Expected
equals	Dininghall.equals(diningHallSame)	True
Not equals	diningHall.equals(kitchen)	False

Testing: ToString	Input	Expected
String representation of constructor	SpaceImpl("Corto Maltese", 2, 21, 13, 24, 18)	"Corto Maltese;2;21;13;24;18"

Testing design for ItemImpl:

Testing: getNameOfItem()	Input	Expected
Get the item name	getNameOfItem()	“Baseball Bat”

Testing: isItemRemoved()	Input	Expected
Item has been used and is evidence		True
Item has not been used in attack		False
Item has not been picked up		False
Item is with player		False

Testing: getSpaceIndexOfItem()	Input	Expected
Get the room location	getSpaceINdexOfItem(“Baseball Bat”, 3, 1)	1

Testing: Constructor	Input	Expected
Valid item	ItemImpl(“Baseball Bat”, 3, 1)	“Baseball Bat; 3; 1”
Invalid damage	ItemImpl(“Baseball Bat”, 0, 1)	IllegalArgumentException
Invalid damage negative	ItemImpl(“Baseball Bat”, -3, 1)	IllegalArgumentException
Invalid room location	ItemImpl(“Baseball Bat”, 3, -1)	IllegalArgumentException
Invalid item name	ItemImpl(“”, 3, 1)	IllegalArgumentException
Invalid item name null	ItemImpl(null, 3, 1)	IllegalArgumentException
Invalid damage and room location	ItemImpl(“Baseball Bat”, -3, -1)	IllegalArgumentException
Invalid item name and room location	ItemImpl(“”, 3, -1)	IllegalArgumentException
Invalid name, damage, and room location	ItemImpl(“”, -3, -1)	IllegalArgumentException
Invalid health and name	ItemImpl(“Baseball Bat”, -3, 1)	IllegalArgumentException

Testing: Equals	Input	Expected
Same object	ItemImpl(“Baseball Bat”, 3, 1)	True
Different objects with same values	ItemImpl(“Baseball Bat”, 3, 1)	True

Different objects; different values	ItemImpl(“Baseball Bat”, 3, 1)
	ItemImpl(“Machine Gun”, 3, 1)
		False

Testing: Hashcode	Input	Expected
Same object	ItemImpl(“Baseball Bat”, 3, 1)	Assert.equals(item1.hashcode(), item2.hashcode()) -> true

Different	ItemImpl(“Baseball Bat”, 3, 1)
	ItemImpl(“Baseball Bat”, 2, 3)
		Assert.equals(item1.hashcode(), item2.hashcode()) -> false

Testing: ToString	Input	Expected
Get string value of item	toString()	“BaseballBat;3;1”

Testing design for Target:

Testing: Constructor	Input	Expected
Valid target	Target(“Starro”, 9, 50)	“Starro;50:0;9”
Invalid health	Target(“Starro”, 9, 0)	IllegalArgumentException
Invalid Blank name	Target(“”, 9, 50)	IllegalArgumentException
Invalid name is null	Target(null, 9, 50)	IllegalArgumentException
Invalid max number of spaces	Target(“Starro”, 0, 50)	IllegalArgumentException

Invalid health and max spaces	Target(“Starro’”, 0, -50)	IllegalArgumentException
Invalid name and max spaces	Target(“”, -1, 50)	IllegalArgumentException
Invalid health and name and spaces	Target(“”, 0, -50)	IllegalArgumentException
Invalid name and health	Target(“”, 9, -50)	IllegalArgumentException

Testing: getTargetName()	Input	Expected
Get the name	getTargetName()	“Starro”

Testing: getCurrentSpace()	Input	Expected
Get the room index	getCurrentSpace()	0

Testing: getCurrentHealth()	Input	Expected
Get the target’s health	getCurrentHealth()	50

Testing: setCurrentHealth()	Input	Expected
Check health and then Set target’s health	setCurrentHealth(5)	Assert should be 50 and then it should be 5
Set target health negative returns 0	setCurrentHealth(-5)	0

Testing: toString()	Input	Expected
Get string representation of Target	toString()	“Starro;50;0;9”

Testing: Move()	Input	Expected
Target moves to next room index from start point	moveTarget()	Assert 0 and then after move assert 1

Check target location at the beginning and then move it past last room back to beginning again	moveTarget() nine times	Assert) and then assert 0 again.

Notes

It is my intention that over spring break I am going to sit back and re-evaluate how I have constructed my design. I want to reflect and hopefully integrate more of the topics that we have covered since milestone one. I feel that there are a number of aspects that can be revisited to improve the overall design. Thus, I expect that by the submission of milestone three it will look very different than this.