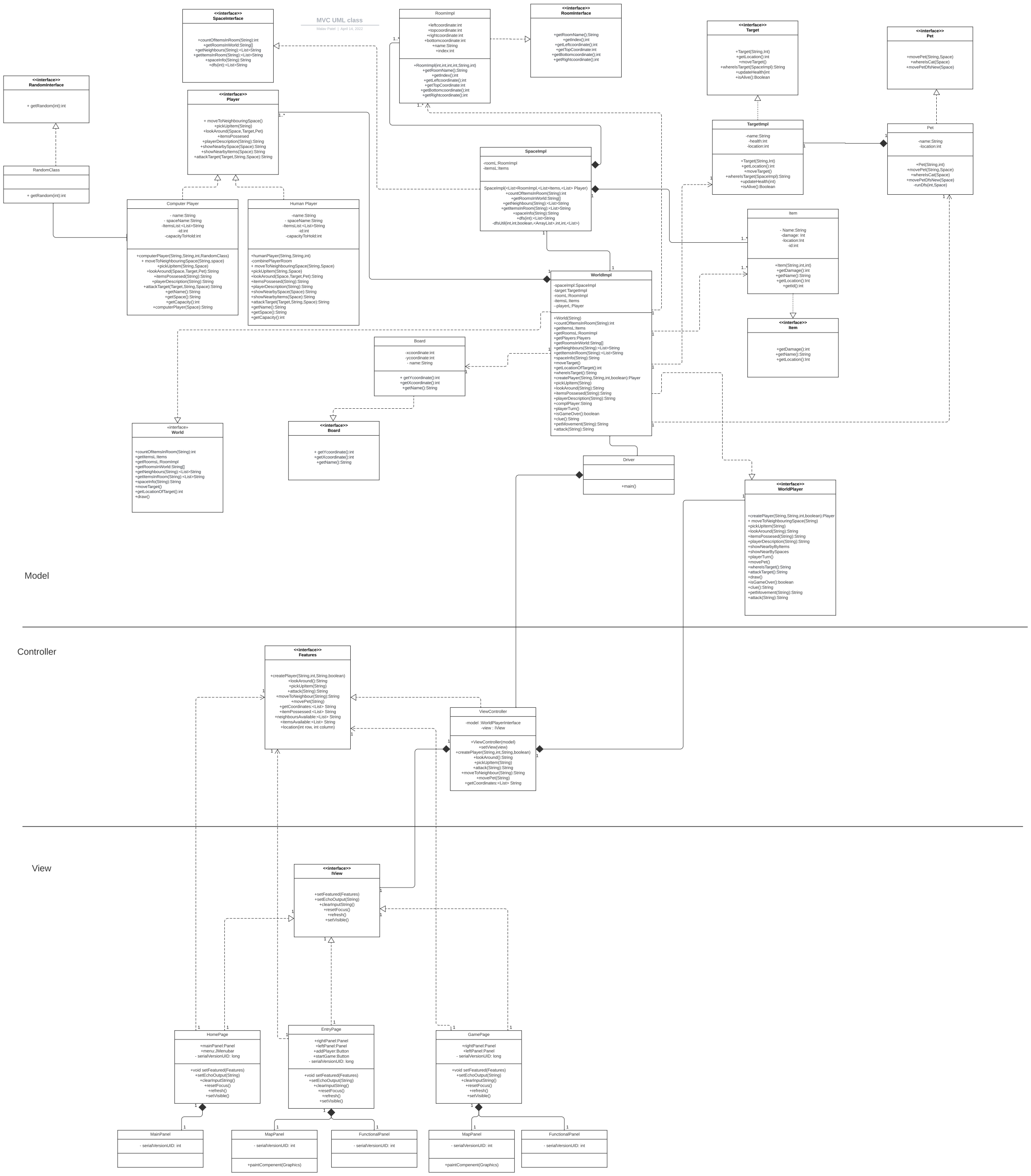
MVC UML class

Malav Patel | April 14, 2022



Tests for Milestone 1

1. **Tests for Driver class**
   - Throw exception if the file does not exist or the wrong location specified.
   - Throw an exception if the file is empty.
2. **Tests for Board class**
   1. Board (int, int, String)

      Constraints – minimum value of both the integers should be greater than or equal to 30.

Testing Constructor	Input	Expected Output
Normal Case	Board (30,30,"MyWorld")	Object created
X coordinate is 0	Board (0,30,"MyWorld")	IllegalArgumentException
Y coordinate is 0	Board(30,0,"MyWorld")	IllegalArgumentException
X coordinate is negative	Board(-30,30,"MyWorld")	IllegalArgumentException
Y coordinate is negative	Board(-30,30,"MyWorld")	IllegalArgumentException
X coordinate is 0 and Y coordinate is negative	Board(0,-30,"MyWorld")	IllegalArgumentException
X coordinate is negative and Y coordinate is 0	Board(-30,0,"MyWorld")	IllegalArgumentException
One of the coordinate is less than 30	Board(20,30,"MyWorld")	IllegalArgumentException

2. **getAllSpace()**

   This function would return a list of all spaces. Verify that the list of spaces returned is the same as the ones mentioned in the input file.

Test getAllSpace	Expected Output	
getAllSpace()	List<Space>	It returns list of all spaces

3. **getNumberOfSpaces()**

   This function would return the number of spaces. Verify that number of spaces returned equals the total number of spaces listed in the input text file.

Test getNumberOfSpaces	Expected Output	
getNumberOfSpaces()	25	

4. **getAllItems()**

   This function returns the list of all available items. Verify that the list of items returned is the same as the ones mentioned in the input file.

Test getAllItems	Expected Output	
getAllItems	<List> Items	It returns list of all items specified in the text file

5. **getNumberOfItems()**

   This function would return the number of Items. Verify that number of items equals the total number of items listed in the input text file.

Test getNumberOfItems	Expected Output	
getNumberOfItems()	25	Assuming that there are 25 items in the game

3. Test for Room Class

1. **Room (int, int, int, int, String, int)**

The space constructor holds values of left, top, right, and bottom coordinates of

Space object accordingly. Certain rules that should be followed are:

- left < right
- top < bottom
- Test whether any of them is not negative, else throw an exception
- They should not overlap with coordinates of other spaces

The 5th variable in the string constructor is also an int which represents the index

- Test whether the index is not larger than the total spaces available
- Test whether the index is not negative.

Testing Space Constructor	Input	Expected Output
Valid Inputs	Room (10,15,30,25,"Armory",0)	Object created
Left coordinate > Right Coordinate	Room (40,15,30,25,"Armory",0)	Illegal Argument Exception
Top coordinate > Bottom Coordinate	Room (10,35,30,25,"Armory",0)	Illegal Argument Exception
Index < 0	Room (10,15,30,25,"Armory",-2)	Illegal Argument Exception
Index > total number of available spaces (assume total number of spaces = 22)	Room (10,15,30,25,"Armory",26)	Illegal Argument Exception

Left coordinate is negative	Room (-10,15,30,25,"Armory",0)	Illegal Argument Exception
Right coordinate is negative	Room (10,15,-30,25,"Armory",0)	Illegal Argument Exception
Top coordinate is negative	Room (10,-15,30,25,"Armory",0)	Illegal Argument Exception
Bottom coordinate is negative	Room (10,-15,30,-25,"Armory",0)	Illegal Argument Exception

4. Test for SpaceImpl class

1. SpaceImpl(List<Rooms>,List<Items>)

SpaceImpl class takes a list of all room objects and list of all items objects and perform important functions on them.

2. getNeighbours(Space)

This method takes in space object and return the list of spaces visible from that space. All the visible places are neighbours.

- There are 4 conditions of a neighbouring space –
  - The left coordinate is one less than the space object passed and all other parameters are valid
  - The top coordinate is one less than the space object passed and all other parameters are valid
  - The bottom coordinate is one more than the space object passed and all other parameters are valid
  - The right coordinate is one more than the space object passed and all other parameters are valid

  So all the other spaces which don't have any one of these types of coordinates should not be included in the list returned.

- Test whether the space passed in the method does exist on the board else throw an exception.
-

Test getNeighbours (Space)	Expected Output	
getVisible(Space)	String eg. Kitchen, Bathroom, Cellar	It returns list of all spaces adjacent to the space passed as the parameter

3. getRoomDetails(String)

This method is similar to toString method. It returns all the visible spaces from the room and all the items present in the room.

- Use assert equals to test the string returned by the getRoomDetails() against a manually typed string.

Test getRoomDetails()	Expected Output	
getRoomDetails()	Armory has Kitchen, Cellar and Bathroom as its neighbour. It has a gun in it.	It returns the name of the space, its neighbours, and items present.

4. countOfItemsInRoom(String) Roomname is passed as paramter

Test countOfItemsInRoom (String)	Expected Output	
countOfItemsInRoom (String)	3	It returns the number of Items in a space.

5. Test for the target class

1. **Target(String,int)**

- Check whether the integer passed is a non-negative value as health can never be negative.
- Check whether integer passed is not zero

Testing Target Constructor	Input	Expected Outcome
Valid Constructors	Target(“Dr. Pyscho”,20)	Object created
Health is zero	Target(“Dr. Pyscho”,20)	Illegal Argument Exception
Health cannot be negative	Target(“Dr. Pyscho”,20)	Illegal Argument Exception

2. **getLocation()**

- Test this method before the moveTarget() has been called even once and it should return 0.

- The value returned by it should always be less than or equal to the total number of spaces available.

- It can be tested by using moveTarget(). Every time after the using the moveTarget() ,the value returned by getLocation() should be incremented by 1.

- Throw Illegal State Exception if value return is negative or more than total spaces available.

Test getLocation()	Expected Output	
getLocation()	0	Assuming that the target has made no move till now.
getLocation()	5	Assuming that the target has made 5 moves till now.

3. **moveTarget()**

- This method moves the target from one index to its next index. Use getLocation() before and after using this method and the difference in answer should be 1.

-This method should throw an Illegal State exception when the target is already at the last index position.

Test moveTarget()	Expected Output	
moveTarget()	1	Assuming that the target has made its first move. In order to test this we need to call getLocation().
moveTarget()	5	Assuming that the target has made its first move. In order to test this we need to call getLocation().

4. Test for the Item class

1. **Item(String, Int, Int)**

The Item constructor takes Name in form String, Damage in form of int and location in form of int.

- Test that neither of the integers should be negative.

- Test that the value of location integer can never be larger than the total number of spaces that exist.

- Test that Damage can never be zero.

Testing Weapon Constructor	Input	Expected Output
Valid Case	(“Gun”,5,0)	Object created
Damage is negative	(“Gun”,-5,0)	Illegal Argument Exception
Index of location is negative	(“Gun”,5,-7)	Illegal Argument Exception
Index of location greater than the total number of spaces possible (Assume spaces = 22)	(“Gun”,5,23)	Illegal Argument Exception
Index of location greater than the total number of spaces possible (Assume spaces = 22)	(“Gun”,5,22)	Illegal Argument Exception (index start from 0)
Damage and Index both are negative	(“Gun”,-5,-7)	Illegal Argument Exception

2. **getDamage()**

-Test whether the integer returned, is similar to the damage mentioned in the input file.

Test getDamage()	Expected Output	
getDamage()	5	This gives the information about the amount of damage.

3. **getName()**

- Test whether the string returned, is similar to the string mentioned in the input file for that particular space.

Test getName()	Expected Output	
getName()	Knife	This returns the name of the item.

4. **getLocation()**

- Test whether the index returned, is similar to the index mentioned in the input file for that particular space.

Test getLocation()	Expected Output	

getLocation( )	Armory	Assuming that the specific item it present in the Armory.

Test Cases for Milestone 2

5. **Test for computerPlayer class**

1. computerPlayer(String,String,int)

The computerPlayer takes in Name in form of String, room name in form of String and capacity to carry in form of int.

- The capacity to carry can never be zero or a negative number.

- Test that name of player is not repeated.

- Test that name of location passed exists.

Testing Constructor	Input	Expected Output
Normal Case	computerPlayer(Malav,Armory,5)	Object created
Item carrying capacity is negative	computerPlayer(Malav,Armory,-5)	Illegal Argument Exception
Item carrying capacity is zero	computerPlayer(Malav,Armory,0)	Illegal Argument Exception
Name is not unique (because according to specification file , all players are recognized by their names)	computerPlayer(Malav,Kitchen, 3)	Illegal Argument Exception
Item carrying capacity exceeds (Assumption – max capacity = 5)	computerPlayer(Malav,Armory,500)	Illegal Argument Exception
Initializing player on a location that doesn’t exist	computerPlayer(Malav,War room,4)	Illegal Argument Exception

2. moveToNeighbour(String)

The parameter passed will give information about which room the player will move next to.

Test moveToNeighbour(String)	Input	Output
Normal Case	moveToNeighbour(Malav,Kitchen)	Player moves
Name does not exist	moveToNeighbour(Henry, Kitchen)	Illegal Argument Exception
Location does not exist	moveToNeighbour(Henry, War Room)	Illegal Argument Exception

3. pickUpItem(String)

Here item name will be passed as a parameter which will give information about which item to pick up.

Test pickUpItem (String)	Input
Normal Case	pickUpItem(Malav,Bottle Opener)
Name does not exist	pickUpItem (Henry,Revolver)
Item carrying capacity is full	pickUpItem(Malav,Revolver)

4. lookAround(String)

Here the Name of player will be passed as an argument

Test lookAround(String)	Input
Normal Case	lookAround(Malav)

Name does not exist	lookAround(Henry)	

5. itemsPossesed(String)
Here the name of the player will be passed as an argument

Test itemsPossesed	Input	
Normal Case	itemsPossesed(Malav)	

Name does not exist	itemsPossesed (Henry)		

6.Test for humanPlayer class

1. humanPlayer(String,String,int)

The humanPlayer takes in Name in form of String, room name in form of String and capacity to carry in form of int.

- The capacity to carry can never be zero or a negative number.

- Test that name of player is not repeated .

- Test that name of location passed exists.

Testing Constructor	Input	Expected Output
Normal Case	humanPlayer(Malav,Armory,5)	Object created
Item carrying capacity is negative	humanPlayer(Malav,Armory,-5)	Illegal Argument Exception
Item carrying capacity is	humanPlayer(Malav,Armory,0)	Illegal Argument

zero		Exception
Item carrying capacity exceeds (Assumption – max capacity = 5)	humanPlayer(Malav,Armory,500)	Illegal Argument Exception

2. moveToNeighbour(String)

The parameter passed will give information about which room the player will move next to.

Test moveToNeighbour(String)	Input	Output
Normal Case	moveToNeighbour(Malav,Kitchen)	Player moves
Name does not exist	moveToNeighbour(Henry, Kitchen)	Illegal Argument Exception
Location does not exist	moveToNeighbour(Henry, War Room)	Illegal Argument Exception

3. pickUpItem(String)

Here item name will be passed as a parameter which will give information about which item to pick up.

Test pickUpItem(String)	Input	Output
Normal Case	pickUpItem(Malav,Bottle Opener)	Player moves
Name does not exist	pickUpItem (Henry,Revolver)	Illegal Argument Exception
Item carrying capacity is full	pickUpItem(Malav,Revolver)	Illegal Argument Exception

4. lookAround(String)

Here the Name of player will be passed as an argument

Test lookAround	Input	Output
Normal Case	lookAround(Malav)	String output giving information about where other players are present and what spaces are visible to them
Name does not exist	lookAround(Henry)	Illegal Argument Exception

5. itemsPossesed(String)

Here the name of the player will be passed as an argument

Test itemsPossesed	Input	Output
Normal Case	itemsPossesed(Malav)	String output giving information about items

		in possession
Name does not exist	itemsPossesed (Henry)	Illegal Argument

		Exception

Adding or Modifying methods defined in classes for Milestone 1

Class - WorldImpl
Method – createPlayer(String,String,int,boolean)
1st parameter defines the name of the player by which it will be identified.
2nd parameter defines the room in which player will start the game.
3rd parameter defines the capacity of the player to carry the items.
1 item = 1 capacity occupied
4th parameter defines what type of player it is.
True = human player
False = computer player

Test createPlayer()	Method calls	Output
Normal case for a computer player	createPlayer(Malav,Armory,5,False)	Create an object of a computer player
Normal case for a human player	createPlayer(Malav,Armory,5,True)	Create an object of a human player
Initializing human player on a location that doesn't exist	createPlayer(Malav,War room,4,True)	Illegal Argument Exception
Initializing computer player on a location that doesn't exist	createPlayer(Malav,War room,4,False)	Illegal Argument Exception
Human Player Name is not unique (because according to specification file , all players are recognized by their names)	humanPlayer(Malav,Kitchen, 3)	Illegal Argument Exception

Method – turn()

This method will assign turns to the players. The order of assigning will be similar to the order in which they were added. This method will be tested by getTurn() which returns the nameof the player whose turn it is.

Class – SpaceImpl
Method – spaceInfo(String)
Roomname is passed as a parameter to this method

Test spaceInfo(String)	Method calls	Output
Normal Case	spaceInfo(Armory)	Gives information about its neighbours, items present and players present in it.
Invalid Room name	spaceInfo(Bar)	Illegal Argument exception

Class – Target
Method – moveTarget()

Target will move to the next indexed location when turn() is invoked . This movement of target can be validated using getLocation() and whereIsTarget().

In order to test whether the controller passes the right value to the model, we need to test the controller in isolation.

Testing the controller

1. Testing move(String) returns void

   A mock model with private StringBuilder log and int testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When move(String) function is invoked, the StringBuilder log will log the function and input parameter Room name which is of type string and testCode will log which test is testing this mock model. In test, the test compares the StringBuilder and testCode with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

   For eg. For test number 1 with playGame(mock) with input "Armory" the expected value of mock's log would be "move(Armory) test 1".

Test move()	Input	Expected
Normal Case	Armory	move(Armory) test 1

2. Testing pickupItem(String) returns void

   A mock model with private StringBuilder log and int testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When pickUpItem (String) function is invoked, the StringBuilder log will log the function and input parameter Room name which is of type String and testCode will log which test is testing this mock model. In test, the test compares the StringBuilder and testCode with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

   For eg. For test number 1 with playGame(mock) with input "Gun" the expected value of mock's log would be "pickUpItem(Gun) test 1".

3. Testing lookAround(String) returns String

   A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When lookAround(String) function is invoked,the StringBuilder log will log the input parameter – Player Name which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. One assertEquals would check the input returned by string builder log while other assertEquals would verify the returned testCode with the expected.

4. Testing createPlayer(String,String,int,boolean) returns void

A mock model with private StringBuilder log and int testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When createPlayer (String,String,int,boolean) function is invoked, the StringBuilder log will log the function and input parameters and testCode will log which test is testing this mock model. In test, the test compares the StringBuilder and testCode with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. For test number 1 with playGame(mock) with input "Malav,Armory,1,True" the expected value of mock's log would be "createPlayer(Malav,Armory,1,True) test 1".

Test createPlayer()	Input	Expected
Normal Case	Malav,Armory,1,True	createPlayer(Malav,Armory,1,True) test 1

Testing spaceInformation(String) returns String

A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When spaceInformation(String) function is invoked,the StringBuilder log will log the input parameter – Room Name which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. One assertEquals would check the input returned by string builder log while other assertEquals would verify the returned testCode with the expected.

Testing playerDescription(String) returns String

A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and a int params such as WorldImplMock (StringBuilder log, int testCode). When playerDescription(String) function is invoked,the StringBuilder log will log the input parameter – Room Name which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. One AssertEquals would check the input returned by String builder log while other AssertEquals would verify the returned testCode with the expected.

Test Cases for Milestone 3

7. Test case for cat class

   1. Pet constructor

The pet constructor will take the name of the pet and location index as an input.

Testing Constructor	Input	Expected Output
Normal Case	Pet (Fortune,0)	Object created

2. movePet(String)

Here the name of the room where the player wants the pet to be moved will be given as an argument

Test movePet(String)	Method calls	Output
Normal Case (Armory is not neighbouring room of the pet’s current room)	movePet (Armory)	Moves the cat to the specified location.
Invalid Room name	movePet (Bar)	Illegal Argument exception
Normal Case (Foyer is not neighbouring room of the pet’s current room)	movePet (Foyer)	Moves the cat to the specified location

3. whereIsPet(space)

This method takes in the space object as a parameter and gives information about the space where the pet is present.

Test whereIsPet(space)	Method calls	Expected Output
Normal Case	whereIsPet(space)	Gives the name of the space where the pet is present.

Additional/Modified methods added to HumanPlayer class

1. attackTarget(Target)

This method can be used when the player and target are in the same space.

Test attackTarget()	Method calls	Output
Normal Case (No other player in neighbouring space , player has only one weapon)	attackTarget(Target)	Health of target is reduced by the amount equal to damage amount of weapon used. Info about updated health is given as output.
Normal Case (No other player in neighbouring space , player has multiple weapons)	attackTarget(Target)	Health of target is reduced by the amount equal to maximum damage caused from available weapons used. Info about updated health is given as

		output.

Normal Case (No other player in neighbouring space , player has no weapon)	attackTarget(Target)	Health of target is reduced by one only. Info about updated health is given as output.
Fail Case (Other player in neighbouring space , player has no weapon)	attackTarget(Target)	Health of the target remains the same. Info about attack failure is given as output.
Fail Case (Other player in neighbouring space , player has a weapon)	attackTarget(Target)	Health of the target remains the same and that weapon is removed from player possession and available weapons list and cannot be used. Info about attack failure is given as output.
Fail Case (Other player in same space , player has a weapon)	attackTarget(Target)	Health of the target remains the same and that weapon is removed from player possession and available weapons list and cannot be used. Info about attack failure is given as output.
Fail Case (Other player in same space , player has no weapon)	attackTarget(Target)	Health of the target remains the same and the attack fails. Info about attack failure is given as output.

2. lookAround(String , space)

This method takes a name of player and space object as an argument.

This method gives name of players present around the command invoking player including the list of neighbouring rooms available to move.

Test lookAround	Input	Output
Normal Case	lookAround(Malav,space)	String output giving information about which items are available in the room and its neighbours,where other players are present and what spaces are visible to them and target if its around or in same space.
Normal Case (Pet is present in the neighbouring room)	lookAround(Malav,space)	String output giving information about which items are available in the room and its neighbours,presence of other players in neighbouring spaces and target if its around or in same space. While not

		giving info of other player and space where

		the pet is present.
Name does not exist	lookAround(Henry,space)	Illegal Argument Exception
Normal Case (Pet and target in neighbouring room)	lookAround(Malav,space)	String output giving information about which items are available in the room and its neighbours, the presence of other players in neighbouring spaces. While not giving info of other player ,target and space where the pet is present.

3. showNearbySpace(space)

This method takes in the space object as a parameter. This method is used to guide the player about the available spaces to move.

Test showNearbySpace(space)	Input	Expected Output
Normal Case	showNearbySpace(space)	List of visible neighbours
Normal Case (Pet is in one the neighbouring space)	showNearbySpace(space)	List of visible neighbours except the neighbour in which the pet is present.

4. showAvailableItems(space)

This method takes in the space object as a parameter. This method is used to guide the player about the available spaces to move.

Test showNearbySpace(space)	Input	Expected Output
Normal Case	showNearbyItems(space)	List of available items to pick
Normal Case (after a weapon from that space has been picked up a player)	showNearbyItems(space)	List of available items to pick excluding the item that has been already picked

Additional/Modified methods added to ComputerPlayer class

1. attackTarget(Target)

This method can be used when the player and target are in the same space. The computer player will automatically attack the target player when they are in the same space.

Test attackTarget()	Method calls	Output
Normal Case (No other player in	attackTarget(Target)	Health of target is reduced by the amount equal to damage

neighbouring space , player has only one weapon)		amount of weapon used. Info about updated health is given

		as output.
Normal Case (No other player in neighbouring space , player has multiple weapons)	attackTarget(Target)	Health of target is reduced by the amount equal to maximum damage caused from available weapons used. Info about updated health is given as output.
Normal Case (No other player in neighbouring space , player has no weapon)	attackTarget(Target)	Health of target is reduced by one only. Info about updated health is given as output.
Fail Case (Other player in neighbouring space , player has no weapon)	attackTarget(Target)	Health of the target remains the same. Info about attack failure is given as output.
Fail Case (Other player in neighbouring space , player has a weapon)	attackTarget(Target)	Health of the target remains the same and that weapon is removed from player possession and available weapons list and cannot be used. Info about attack failure is given as output.
Fail Case (Other player in same space , player has a weapon)	attackTarget(Target)	Health of the target remains the same and that weapon is removed from player possession and available weapons list and cannot be used. Info about attack failure is given as output.
Fail Case (Other player in same space , player has no weapon)	attackTarget(Target)	Health of the target remains the same and the attack fails. Info about attack failure is given as output.

2. lookAround(String , space)

This method takes a name of computer player and space object as an argument.

This method gives name of players present around the command invoking player including the list of neighbouring rooms available to move.

Test lookAround	Input	Output
Normal Case	lookAround(Malav,space)	String output giving information about which items are available in the room and its neighbours,where other players are present and what spaces are visible to them
Normal Case (Pet is present in the neighbouring room)	lookAround(Malav,space)	String output giving information about which items are available in the

		room and its neighbours,presence of

		other players in neighbouring spaces while not giving info of other player and space where the pet is present.
Normal Case (Pet and target in neighbouring room)	lookAround(Malav,space)	String output giving information about which items are available in the room and its neighbours, the presence of other players in neighbouring spaces. While not giving info of other player ,target and space where the pet is present.

Additional/Modified methods in WorldImpl class

1. isGameRunning()

It returns false if health of target is zero or number of turns equals to total number of requested turns.

Test isGameOver()	Method calls	Output
Normal Case (There are turns left but target health is zero)	isGameRunning()	Return false which would result in ending the game.
Normal Case (No turns left but target is alive)	isGameRunning()	Return false which would result in ending the game.
Normal Case (There are turns left and target is alive)	isGameRunning()	Return true which would allow the game to continue

Additional/Modified methods added to Controller

1. attackTarget()

For the controller, we need to perform two types of tests.

i. Checking if the right method is invoked using Mocking

A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and a int params such as WorldPlayerImplMock (StringBuilder log, String testCode). When a command attack is passed as an input , attack() function of the mock model is invoked,the StringBuilder log will log the input parameter i.e attack command which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

ii. Checking if the desired output is received by invoking the command

In this type of testing a real model will be used. When attack command is passed as input , it will compare the output of this command with the desired output. By this way we can test whether a specific command is working correctly.

2. movePet()

For the controller, we need to perform two types of tests.

i. Checking if the right method is invoked using Mocking

A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and a int params such as WorldPlayerImplMock (StringBuilder log, String testCode). When a command movePet is passed as an input , movePet() method of the mock model is invoked,the StringBuilder log will log the input parameter i.e attack command which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

ii. Checking if the desired output is received by invoking the command

In this type of testing a real model will be used. When movePet command is passed as input , it will compare the output of this command with the desired output. By this way we can test whether a specific command is working correctly.

3. showNearbyItems()

For the controller, we need to perform two types of tests.

i. Checking if the right method is invoked using Mocking

A mock model with private StringBuilder log and String testCode and its Constructor with a StringBuilder log and an int params such as WorldPlayerImplMock (StringBuilder log, int testCode). When a command showNearbyItem is passed as an input , showNearbyItems() method of the mock model is invoked,the StringBuilder log will log the input parameter i.e attack command which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

ii. Checking if the desired output is received by invoking the command

In this type of testing a real model will be used. When showNearbyItems command is passed as input , it will compare the output of this command with the desired output. By this way we can test whether a specific command is working correctly.

4. showNeighbours()

For the controller, we need to perform two types of tests.

i. Checking if the right method is invoked using Mocking

A mock model with private StringBuilder log and String testCode and its Constructor with a

StringBuilder log and an int params such as WorldPlayerImplMock (StringBuilder log, int testCode). When a command showNeighbour is passed as an input , showNeighbours() method of the mock model is invoked,the StringBuilder log will log the input parameter i.e attack command which is of type String and will return testCode. In test, the test compares the StringBuilder and testCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

ii. Checking if the desired output is received by invoking the command

In this type of testing a real model will be used. When showNeighbours command is passed as input , it will compare the output of this command with the desired output. By this way we can test whether a specific command is working correctly.

Test Cases for Milestone 4

Testing the Controller's interaction Model

1. Testing move(String) returns void

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log, String uniqueCode). When move(String) function is invoked, the StringBuilder log will log the function and input parameter Room name which is of type string will log which test is testing this mock model. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. For uniqueCode “abcTest” with playGame(mock) with input "Armory" the expected value of mock's log would be "move(Armory)”.

Test move()	Input	Expected
Normal Case	Armory	log :move(Armory)

2. Testing pickupItem(String) returns void

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log, String uniqueCode). When pickUpItem (String) function is invoked fromMockWorld, the StringBuilder log will log the function and input parameter Room name which is of type String . In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. For unique code “asdfTest” with playGame(mock) with input "Gun" the expected value of mock's log would be "pickUpItem(Gun)”

Test pickupItem()	Input	Expected
Normal Case	Gun	log :pickUpItem(Gun)

3. Testing lookAround() returns String

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log, String uniqueCode). When lookAround() function is invoked, It will return uniqueCode. In test, the test compares the uniqueCode with expected code to check if Controller in a specific test has successfully invoked the correct method.

Test lookAround()	Input	Expected
Normal Case	–	log :lookAround()

4. Testing attack(String) return void

A mock model with private StringBuilder log and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log). When a command attack is passed as an input , attack() function of the mock model is invoked,the StringBuilder log will log the input parameter i.e attack command which is of type String. In test, the test compares the StringBuilder log with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

Test attack()	Input	Expected
Normal Case	Gun	log :attack(Gun)

5. Testing movePet (String) return void

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log, String uniqueCode). When a command movePet is passed as an input , movePet() method of the mock model is invoked,the StringBuilder log will log the input parameter i.e movepet command which is of type String . In test, the test compares the StringBuilder log with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

Test movePet ()	Input	Expected
Normal Case	Armory	log :movePet(Armory)

6. Testing itemsAvailable() returns List<String>

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and an string params such as MockModel (StringBuilder log, String uniqueCode). When a command itemsAvailable is passed as an input , items() method of the mock model is invoked, It will return uniqueCode. In test, the test compares the StringBuilder and uniqueCode with expected uniqueCode to check if Controller in a specific test has successfully invoked the correct method.

Test itemsAvailable()	Input	Expected
Normal Case	–	log : itemsAvailable()

7. Testing neighboursAvailable() returns List<String>

A mock model with private StringBuilder log and String uniqueCode and its Constructor with StringBuilder log and an String params such as MockWorld (StringBuilder log, string uniqueCode). When the command neighboursAvailable is passed as an input , neighbours() method of the mock model is invoked, It will return uniqueCode. In test, the test compares the StringBuilder and uniqueCode with expected uniquecode to check if Controller in a specific test has successfully invoked the correct method.

Test neighboursAvailable()	Input	Expected
Normal Case		log : neighboursAvailable() uniqueCode:abcTest

8. Testing createPlayer(String,String,int,boolean) returns void

A mock model with private StringBuilder log and string uniqueCode and its Constructor with a StringBuilder log and a string params such as MockWorld (StringBuilder log, String uniqueCode). When createPlayer (String,String,int,boolean) function is invoked, the StringBuilder log will log the function and input parameters . In test, the test compares the StringBuilder with expected input to check if the Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

For eg. For unique code “fdsTest” with playGame(mock) with input "Malav,Armory,1,True" the expected value of mock's log would be "createPlayer(Malav,Armory,1,True)

Test createPlayer()	Input	Expected
Normal Case	Malav,Armory,1,True	Log :createPlayer(Malav,Armory,1,True)

9. Testing itemPossessed() returns List<String>

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and an string params such as MockModel (StringBuilder log, String uniqueCode). When a command itemPossessed is passed as an input , itemsPossessedList() method of the mock model is invoked, It will return uniqueCode. In test, the test compares the StringBuilder and uniqueCode with expected uniqueCode to check if Controller in a specific test has successfully invoked the correct method.

Test itemPossessed()	Input	Expected
Normal Case	–	log :itemPossessed()

		UniqueCode: abcTest

10. Testing getCoordinates() returns List<String>

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and an string params such as MockModel (StringBuilder log, String uniqueCode). When a command getCoordinates is passed as an input , getCoordinates() method of the mock model is invoked, It will return uniqueCode. In test, the test compares the StringBuilder and uniqueCode with expected uniqueCode to check if Controller in a specific test has successfully invoked the correct method.

Test getCoordinates()	Input	Expected
Normal Case	–	log :getCoordinates() UniqueCode: abcTest

11. Testing location(int,int) returns return void

A mock model with private StringBuilder log and String uniqueCode and its Constructor with a StringBuilder log and a String params such as MockWorld (StringBuilder log, String uniqueCode). When a command location is passed as an input , moveCoordinates() method of the mock model is invoked,the StringBuilder log will log the function and input parameter row and col which is of type int and uniqueCode will log which test is testing this mock model. In test, the test compares the StringBuilder and uniqueCode with expected input check if Controller in a specific test has successfully invoked the correct method and passed the correct parameters from user inputs.

Test location(int,int)	Input	Expected
Normal Case	5,6	log :location(5,6)

Testing the Controller's interaction with View

1. Testing setFeatures(Features) returns void

A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When setFeatures(Features) function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

2. Testing setEchoOutput(String) returns void

A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When setEchoOutput(String) function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

3. Testing clearInputString() void

A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When clearInputString() function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

4. Testing resetFocus() returns void

A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When resetFocus() function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

5. Testing refresh() returns void

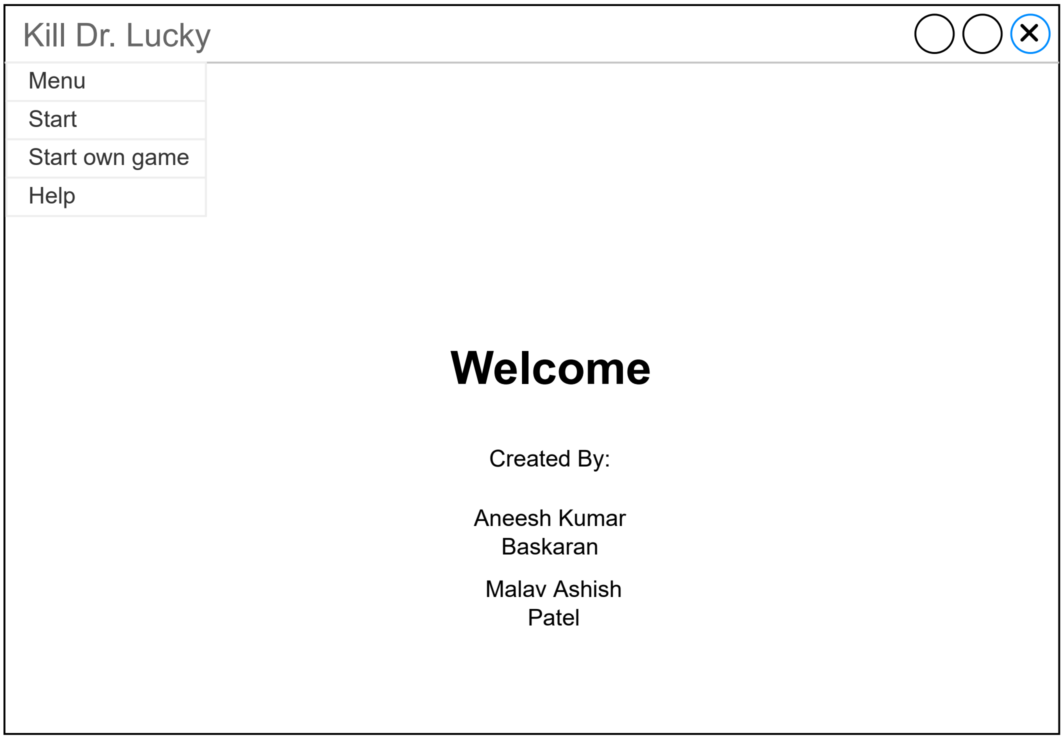
A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When refresh() function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

6. Testing makeVisible() returns void

A mock view with private StringBuilder log and its Constructor with a StringBuilder log such as MockView (StringBuilder log). When makeVisible() function is invoked, the StringBuilder log will log the function. In test, the test compares the StringBuilder log with expected input to check if the Controller in a specific test has successfully invoked the correct method.

Homepage

The starting page of the game



Welcome

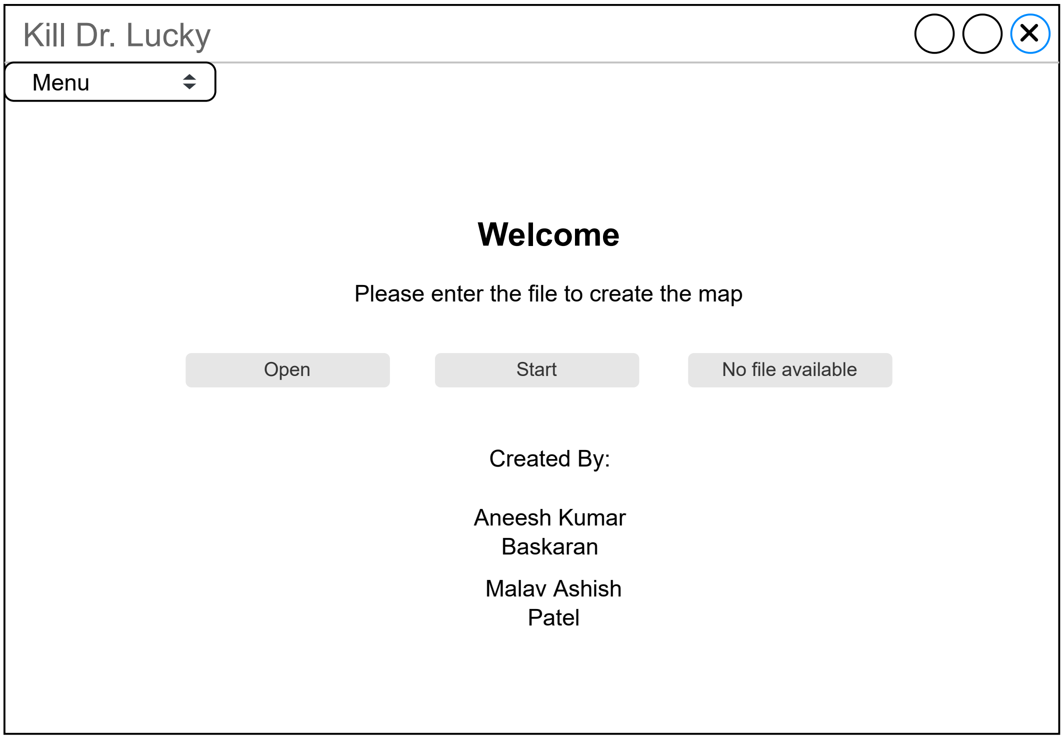
Created By:

Aneesh Kumar
Baskaran

Malav Ashish
Patel

Homepage

If you select "Start own Game"



Welcome

Please enter the file to create the map

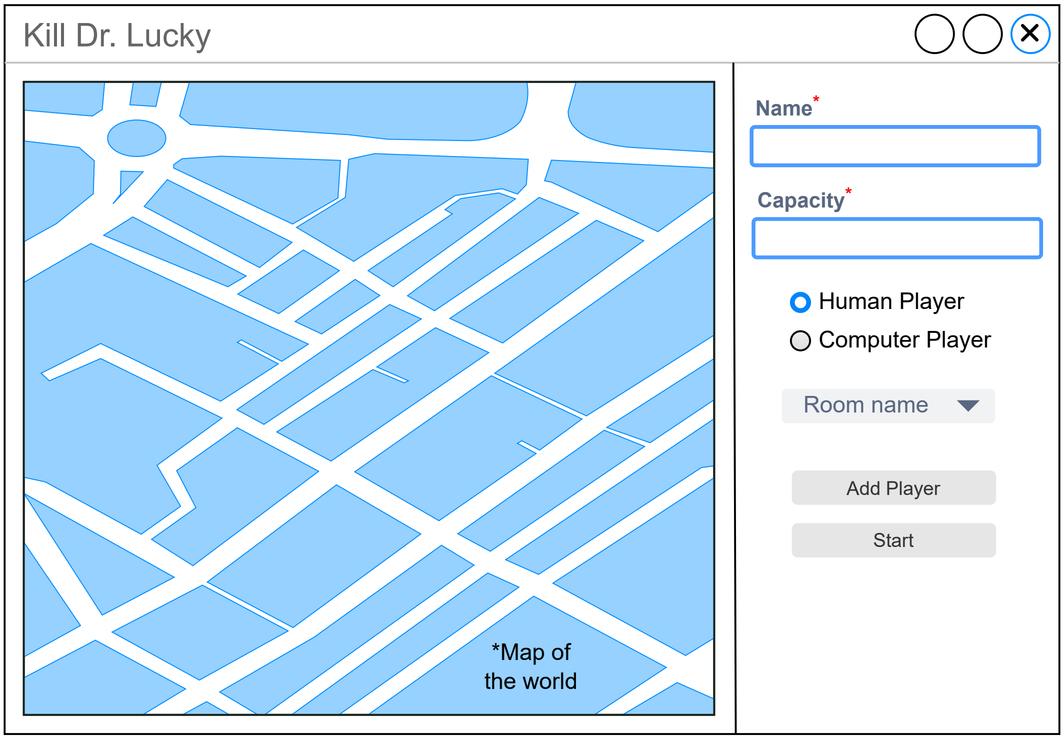
Created By:

Aneesh Kumar
Baskaran

Malav Ashish
Patel

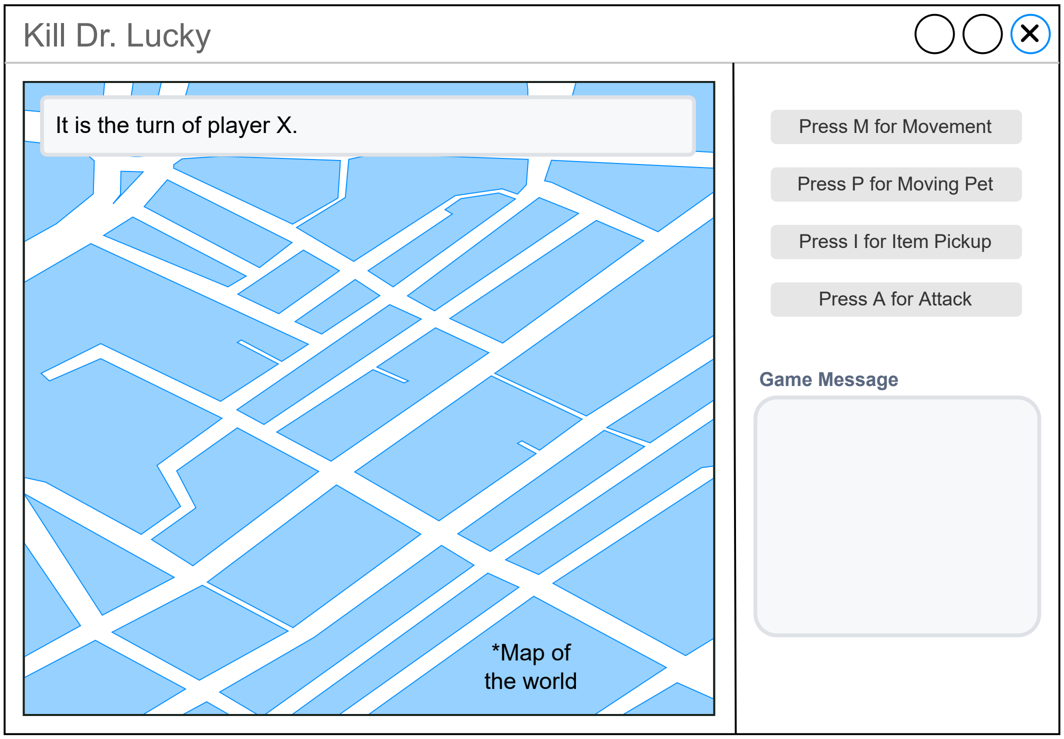
Entry page

After selecting "Start", EntryPage will load



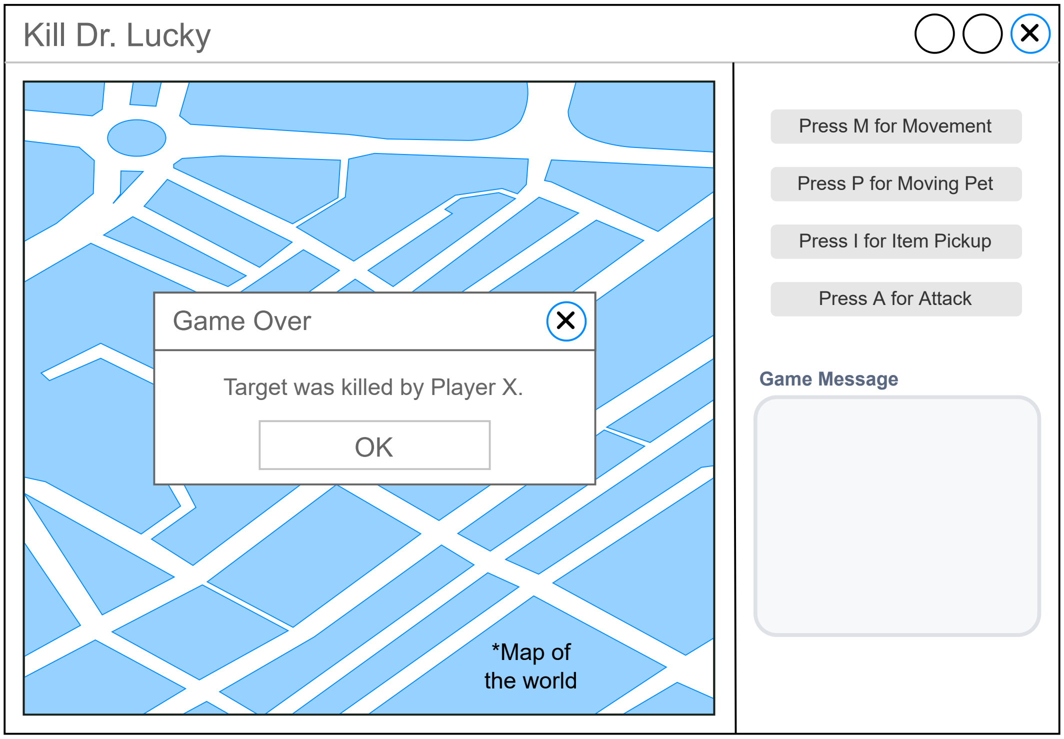
Game Page

After selecting "Start", GamePage will load


It is the turn of player X.

Game Page

Game Over alert pop up



Kill Dr. Lucky

Game Over

Target was killed by Player X.