Kathleen Ann Thurmes
ID# 934039602
CS162: Introduction to Computer Programming II
Final Project

Class Diagram

Menu
Has a Game

Game
Has a GameBoard
Has a Player
Has a Sister
Has a List of NPCs
Has a List of Items

GameBoard
Has a 2D array of Spaces

Space
Has pointers to other Spaces
Has an Item

Character
Has a Space

NPC
Is a Character
Has an Item

Player
Is a Character
Has a list of Items

Sister
Is an NPC

Ghost
Is an NPC

Empty

Is a Space

Fence
Is a Space

Gate
Is a Space

Gravestone
Is a Space

Item

Key
Is an Item

Design Descriptions

Menu
Game game

welcome()
startMenu()
launchGame()
gameOptions()

Game
Player player Sister sister GameBoard gb list<*NPC> NPCList list<*Item*> itemList Bool gameOver

moveCharacter(Character*, Space*)
spawnGhost(Space*)
Bool checkGameOver()
spawnSnack()
playTurn()

GameBoard
Space*** spaces
printGameBoard()

Space
Character* occupier Space* top Space* right Space* bottom Space* left

Bool passable Item* item String symbol
Getters and setters Virtual interactWithPlayer() printSpace()

Character
Space* location String symbol
Virtual move() pickUpItem() Getters and setters

Player: Character
Int health List<Item*> inventory
printHealth() Friend Space::interactWithPlayer() Friend NPC::interactWithPlayer()

NPC: Character
Item* item Int direction

Virtual interactWithPlayer()
dropItem()
step()
turn()
disappear()

Sister: NPC
Bool found

Ghost: NPC

Empty:Space

Fence:Space

Gravestone: Space
Bool hasKey

Gate:Space

Item
String itemName
Getters and setters

Key: Item

Snack:Item

Menu::gameOptions()

```
Cout 1. Play turn
2. View inventory
3. View player health
4. Quit
if(selection ==1){
game.playTurn()
```

Game::playTurn()

```
Player makes move
Call interactWithPlayer() with any NPC they encounter
Update health
Call interactWithPlayer() with any space they encounter
Update gameOver bool
Sister makes move
NPCs make moves (but do not step on newly-moved player)
```

Game::moveCharacter(Space* destination)

```
Check that space is passable and that there isn’t a Character already there
Update Space player is currently on so it doesn’t point to player any more
Update Player’s Space pointer to point to destination
```

```
Destination’s Character pointer points to Character
```

Game::spawnGhost(Space* destination)
```
Check that space is unoccupied and that space is passable
Add Ghost to NPCList
Call moveCharacter to place Ghost
```

Bool Game::checkGameOver()
```
Check if the player’s health has run out (loss)
Check if player found sister, has key, and has exited the cemetery (win)
```

Game::spawnSnack(Space* destination)
```
Check that space is unoccupied and that space is passable
Add Snack to ItemList
Call destination’s setter function with Snack
```

GameBoard:: printGameBoard()
```
Call printSpace() for each space
```

Space::printSpace()
```
Print -----
Print |   |
Print | (insert space’s symbol or Character’s symbol if there’s a Character there) |
Print |   |
Print -----
```

Player::move()
```
Get user input for movement
Check that there’s a space there
Check for NPC
    If there’s an NPC, call interactWithPlayer()
Call interactWithPlayer() on space
Move to space if passable
```

NPC::step()
```
Move one Space in the direction that they’re facing
```

NPC::turn()
```
Use random int to determine whether they’re turning clockwise or counter-clockwise
Use same method that I used in Langton’s Ant
```

NPC::disappear()
```
Remove space’s pointer to NPC
```

Set NPC’s location pointer to null
Do not delete the NPC. The pointer to NPC should still exist in Game’s NPCList or as Game’s Sister object.

Sister::interactWithPlayer()
Allow user to decide to pull sheet off “ghost” or not
When sister is revealed, set “found” to true
Call disappear()

Ghost::interactWithPlayer()
Print something about feeling a cold breeze
Decrease Player’s health

Empty::interactWithPlayer()
{}

Gravestone::interactWithPlayer()
```
If hasKey
    Ask if player wants to clean off and read the epitaph
    If so, discover that it’s a fake gravestone and hides a key
    Set to passable and don’t print gravestone symbol in the future
If !hasKey
    Ask if player wants to wiggle it
    If so, spawn a Ghost
```

Gate::interactWithPlayer()
```
Check if Player has key in inventory
If so, unlock the gate.
Set gate’s “passable” bool to true
If player tries to leave,
    If sister is found
        Set Game’s gameOver bool to true
    Else
        Print message about not being able to go home without sister
```

Fence::interactWithPlayer()
Print something like, “this looks too high to climb, and there are spikes at the top”

Test Tables

It would be awful to write up all the tests I'll have to do on all the smaller components, so this will just focus on testing the final product. Input validation has already been tested, of course.

Thing Being Tested	**Action**	**Expected Results**	**Passed?**
Main Menu allows player to launch a new game	Select "start new game" at beginning	Starts a new game and prints backstory	Y
Main Menu allows player to quit	Select "quit" at beginning	Quits	Y
View backpack contents option works	Select "view backpack contents" on game menu	Prints the contents of the backpack and asks if user would like to use one	Y
Use object recognizes when there are no objects to use and returns user to menu	User chooses to use an item, but doesn't have any	Kick back to an earlier menu	Y
HarryPotter comes back to life	Call defense(int) on Harry Potter many times. Add print statement to show how many dice are being rolled. Add print statement to indicate that HP is coming back to life	Harry Potter should come back to life with strength = 20 only once before he dies for real.	Y
View player health displays player health	Select "view player health" on game menu	Player health displays and it starts at 30	Y
"Make a move" option allows user to choose direction for player to move	Select "make a move option"	Bring user to menu to select which direction to go in	Y

Player’s health is decremented after each turn	Print health, make a move, then print out health	Player’s health is decremented	Y
Moving in any direction works as intended	User selects “up”, “right”, “down,” “left,” and “don’t move”	Player moves to correct space (or doesn’t, if the space is not passable)	Y
Pointers are updated as Characters move around	Player moves onto Spaces previously occupied by NPCs	Game allows player to make move	Y
NPCs move as intended	Observe NPC movement for a few turns	NPCs move about half the time (the other half is turning)	Y
Non-key Gravestone interaction with Player	Allow player to wiggle the headstone then test to see if Player can move through the space once Ghost is gone	Ghost appears, headstone symbol disappears from printout, the space is passable. No key appears.	Y
Key Gravestone Interaction with payer	Allow player to clean off epitaph and read it. Allow player to discover key	Epitaph reads something revealing, player discovers key, gravestone symbol disappears from printout, key appears in printout, player can move onto the key ad pick it up, key appears in inventory, Gravestone Space is passable	Y
Sister interaction with Player	Player bumps into sister	Player pulls off sheet, sister says something, sister disappears, Sister’s “found” bool is set to “true”, sister still exists in Game object, but doesn’t appear in next printout	Y

Ghost interaction with Player	Player bumps into Ghost	Player feels a chill, player can’t move onto destination space (where the Ghost is), player’s health is decreased by 1	Y
Fence interaction with Player	Player bumps into Fence	Print something about it being too high to climb. Player can’t move onto space	Y
Gate interaction with player without key	Player bumps into Gate without Key	Print something about it being too high to climb. Player can’t move onto space	Y
Gate interaction with Player with Key, but no sister	Player bumps into Gate with key	Print something about having to go back for sister. Player can move onto space, but can’t go further	Y
Gate interaction with Player with Key and sister	Player bumps into Gate with Key and having found Sister	Game ends on next turn with celebratory message	Y
Player runs out of health	Player loses 30 health points and doesn’t gain any	Game ends on next turn with boo-hoo message	Y
Ghosts drop snacks	Observe Ghost movement and see that it leaves snacks behind occasionally	Ghost leaves Snacks behind randomly	Y
Snacks Increase Player health	Print Player health. Player picks up a snack and uses it on next turn. Print Player health again.	Player’s health has increased by 5	Y
Snacks disappear from inventory after	Print inventory, use a snack, print inventory	One snack has disappeared from	Y

being used	again	inventory	

Reflection

I made a number of major changes to my code throughout the process of implementing the plan detailed earlier in this document.

First and foremost, I discovered the meaning of the term “circular dependencies.” I found that my Space class could not have a pointer to a Character at the same time that my Character class had a pointer to a Space. Later on, it was mentioned in a Piazza discussion that this was, indeed, possible, but I had already implemented my alternative.

In the end, my Space class became a mess of bools and flags. I had to search through my Game class’ NPCList every time my player landed on a Space that had its hasCharacter flag set to true to find an NPC whose location pointer pointed to the same Space that my Player was trying to move to.

Having to change so that the Space did not contain any information on the Character that was on it ended up meaning that much of the legwork and coordination of the game had to happen in the Game class itself, because that ended up being the central hub of information about what went where. I feel that I did not take advantage of the “Space has pointers to neighboring Spaces” structure as much as I would have if a Space contained all the necessary information to coordinate a turn.

In addition to having to have the Game class get involved with discovering which NPC was on a Space that the Player was landing on, I also ended up having to make the Game class a broker for the interactions between Characters. Originally I thought that I would use friend functions to make the two meeting Character objects simply interact with each other (with the NPC object having access to the Player’s health), but I read up on friend functions, and it seems that friend functions can’t be virtual, so my NPC’s interactWithPlayer() function would not be able to be made a friend. Instead, I chose to have the interactWithPlayer() function return an int that represented the damage that was to be done to the Player object’s health.

Also, printing the game board ended up being a pain and not nearly as simple as what I thought it would be. Again, this would have been much more streamlined had I known that there was a way to avoid circular references. In the end, I had to have the gameBoard object construct a String that would represent all the Spaces and the information that the Spaces had access to. Then, in the Game class, I had a separate function that took the gameBoard’s string and inserted characters into the string to represent the Character objects.

I also created a more user-friendly API around the inventory (which I called the Inventory class). I am still not sure if it was a good idea to implement Inventory using a vector, or if I should have used a list instead. It seemed that both had decent arguments for their use. I ended up going with a vector for its easy access to items using indexes (as opposed to just having access to the front and back items).

There were two features that I added toward the end that were “just for funsies.” First, because I felt that the game was low on Item objects, I added a Sheet Item that the Sister dropped when she disappeared. The sheet could be used to climb over the spiky-topped fence so the user could circumvent getting the key from the Gravestone. I intended to make this option take some health away from the Player, but I did not get around to changing the way the interactWithPlayer() function worked for a Space (I would have had to have the function return an int and made the Game class handle it) or how it was determined whether the game was won or not (if the Player loses the rest of their health while escaping the cemetery, do they win or lose?).

I also added an additional “Glare” feature of the game where the player would lose health just by being in sight of a Ghost NPC. That was just a fun add-on that I did toward the end of the project. I ended up having to adjust how much health the player started with to make the game more win-able.

I also made a helper function to create a pause in the game so the user can read text without having to scroll up past the printed-out game board (enterToContinue). Originally it was just supposed to be “press ‘enter’ to continue”, but I found that introduced a bug in my code that the internet had no idea how to get around. This has to do with clearing the input stream and having to take out the residual ‘\n’ that normally gets left there. So now it’s “type any character and press enter to continue”, which is pretty awkward, but I think it adds enough to the user experience that it is worth it to have in there.

Overall, I appreciated the opportunity to put together a project that showcased many of the things I learned in this class and really emphasized to me how far I’ve come since struggling with the Langton’s Ant assignment. If I were to put this project in my public portfolio, however, I might go back and add those circular references, because I think that would make my code a lot less ugly.