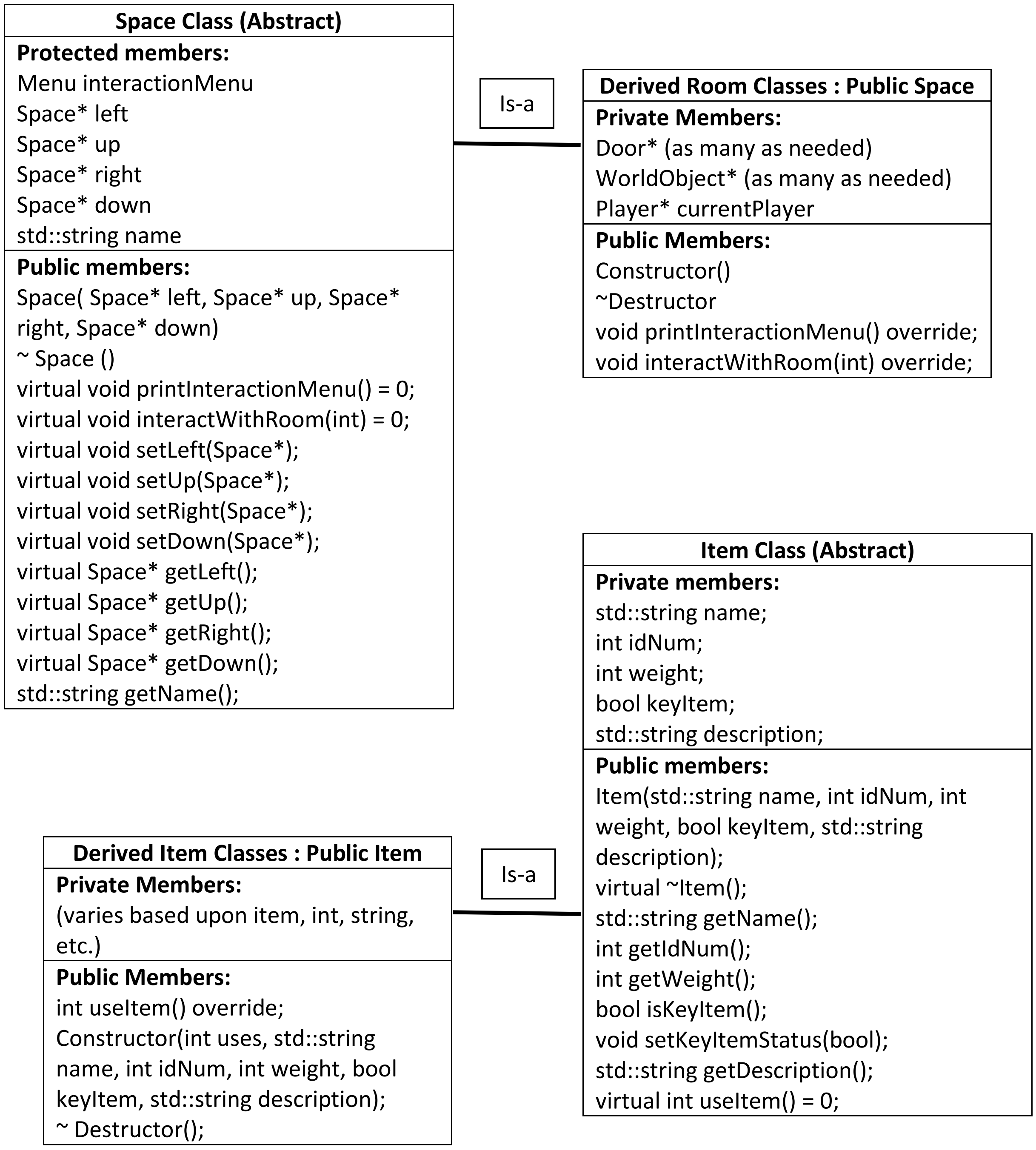
Anthony Clemente
CS162
Final Project Design
13 June 2017
Design:



Space Class (Abstract)

Protected members:
Menu interactionMenu
Space* left
Space* up
Space* right
Space* down
std::string name

Public members:
Space(Space* left, Space* up, Space* right, Space* down)
~ Space ()
virtual void printInteractionMenu() = 0;
virtual void interactWithRoom(int) = 0;
virtual void setLeft(Space*);
virtual void setUp(Space*);
virtual void setRight(Space*);
virtual void setDown(Space*);
virtual Space* getLeft();
virtual Space* getUp();
virtual Space* getRight();
virtual Space* getDown();
std::string getName();

Derived Room Classes : Public Space

Private Members:
Door* (as many as needed)
WorldObject* (as many as needed)
Player* currentPlayer

Public Members:
Constructor()
~Destructor
void printInteractionMenu() override;
void interactWithRoom(int) override;

Item Class (Abstract)

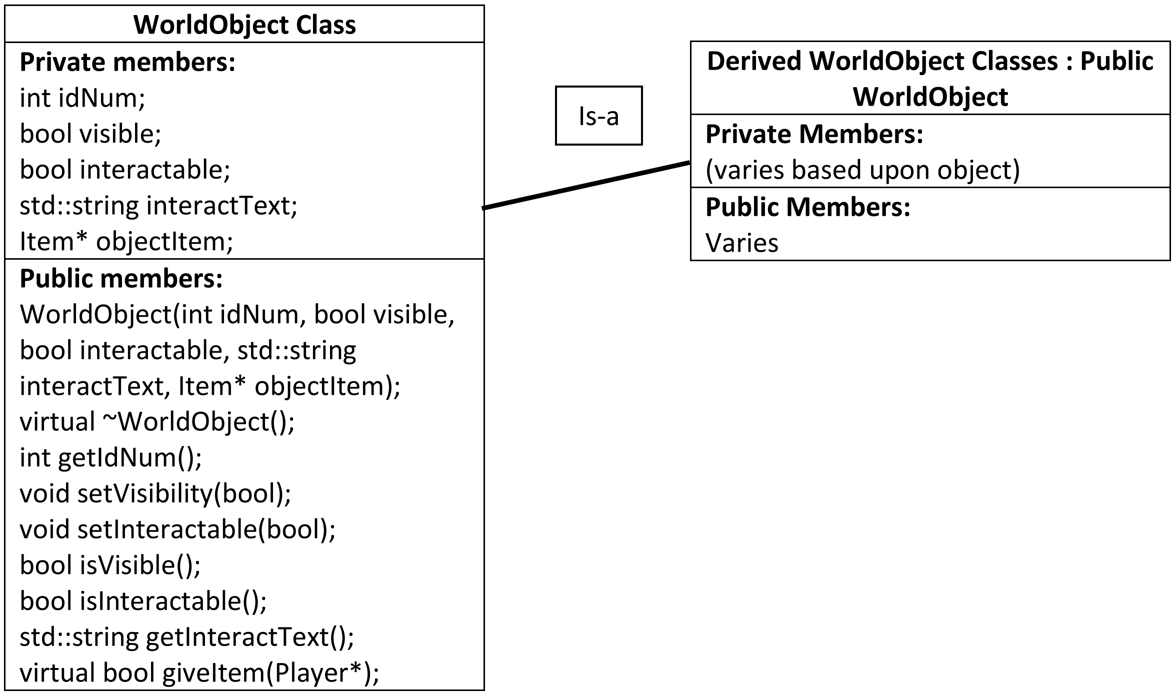
Private members:
std::string name;
int idNum;
int weight;
bool keyItem;
std::string description;

Public members:
Item(std::string name, int idNum, int weight, bool keyItem, std::string description);
virtual ~Item();
std::string getName();
int getIdNum();
int getWeight();
bool isKeyItem();
void setKeyItemStatus(bool);
std::string getDescription();
virtual int useItem() = 0;

Derived Item Classes : Public Item

Private Members:
(varies based upon item, int, string, etc.)

Public Members:
int useItem() override;
Constructor(int uses, std::string name, int idNum, int weight, bool keyItem, std::string description);
~ Destructor();



WorldObject Class

Private members:

```
int idNum;
bool visible;
bool interactable;
std::string interactText;
Item* objectItem;
```

Public members:

```
WorldObject(int idNum, bool visible,
bool interactable, std::string
interactText, Item* objectItem);
virtual ~WorldObject();
int getIdNum();
void setVisibility(bool);
void setInteractable(bool);
bool isVisible();
bool isInteractable();
std::string getInteractText();
virtual bool giveItem(Player*);
```

Is-a

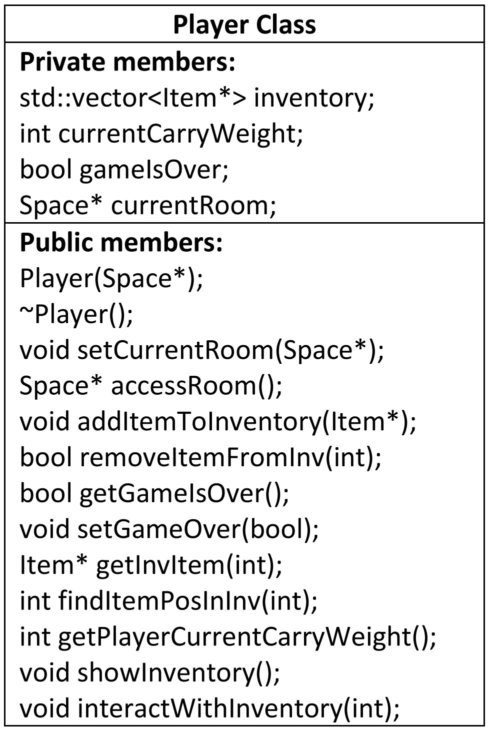
Derived WorldObject Classes : Public WorldObject

Private Members:

(varies based upon object)

Public Members:

Varies



Player Class

Private members:

```
std::vector<Item*> inventory;
int currentCarryWeight;
bool gameIsOver;
Space* currentRoom;
```

Public members:

```
Player(Space*);
~Player();
void setCurrentRoom(Space*);
Space* accessRoom();
void addItemToInventory(Item*);
bool removeItemFromInv(int);
bool getGameIsOver();
void setGameOver(bool);
Item* getInvItem(int);
int findItemPosInInv(int);
int getPlayerCurrentCarryWeight();
void showInventory();
void interactWithInventory(int);
```

The general design of the game focuses on 3 main classes (Space, WorldObject, and Item) with a player class to interact with them. There is an abstract Space class that will be used for derived classes which will represent each room in the house. There will be 7 rooms total: Porch, Living Room, Kitchen, Basement, Dining Room, Bedroom, and Child Room. Each of these classes will derive from the Space class and will be set up in similar ways, but interaction with them will be specific to the room. Rooms will be filled with WorldObjects. WorldObjects are a class that will represent something in the world. They will have an id, an interactable and visible "status" as well as a string to represent what should print when they are "interacted" with. Most importantly WorldObjects will have an Item that can potentially be given to the player under certain circumstances. Every room will have at least one door and the Door class will inherit from the WorldObject class. In the event that a WorldObject, say a door, should not give an Item, this member variable can be made a null pointer. Item will be an abstract class that will be inherited by the number of items that the rooms can offer. Most Items will simply need to display text when interacted with and so will be a derived class TextItem in this case. But there will also be a flashlight, matches, and keys (for doors). The player will have an inventory of items that will be managed by a vector of Item pointers. Each item will have a "useItem" function that will be virtual which will allow for any item to be used in the right way even though they are being accessed through Item pointers.

The player will have a pointer to the room they are currently in and thus will be able to interact with the room through the rooms member function. This member function will have a similar framework for each room but will function specifically to each of the objects each room has. The function will take an integer and then based upon the user choice, a switch statement will execute the appropriate actions. For example, if the player interacts with an object that will potentially give an item, the status of the object can be checked (such as if it is visible and interactable) and then the item can be given if desired. After this the status of the object can be changed (such as setting it as not interactable), for the next time the user chooses to interact with it. The goal of the game will be for the player to collect certain important Items (there are a few not necessary to finish the game). These items will be checked for in the player's inventory by a Ghost class object in the final room. If the proper items are in the inventory, the player wins and the game ends (via the Ghost class changing a boolean member variable of the player)!

The main program will be set up in the following manner. There will be a menu with choices to either interact with the current room, check the player's inventory, inspect an inventory item, or drop an inventory item. Each time after interaction with the room the game will check if the game is over via the player member variable. This only needs to be checked here since after performing an interaction is the only time this can change. The main function will continue to loop until this boolean is set to indicate the game is over.

Testing

After each room is created it will be tested individually to make sure that every interaction works properly (aside from moving to another room). By testing individually this will also fix memory leaks along the way since if there is memory leak I will know it is within the room I am currently testing. If memory leaks occur after putting all the rooms together I will know it is due to something about how I am putting them together if each individual room was good before assembly. After setting up the rooms and the main function at that point it will just be necessary to actually play the game several times to make sure everything is working. Making sure to try and "break" the game will be important. I will try and drop items I don't have and/or enter invalid choices to make sure I have protected against everything I need to. Games can be somewhat notorious for being bug prone since there are so many things interacting with each other. But with a small game like this I should be able to test enough to have pretty good confidence that everything will work how I want it to.

Reflections

This project was a lot of fun to work on and was definitely the biggest thing I put together all term. One thing I can reflect on that is not programming related is that I definitely didn't really craft a very interesting story but instead wanted to focus on actually making the code work properly. Like the assignment description says, the code is the more important part so I am content with that and understand I can always expand upon that later if I desired.

One small thing I noticed while putting together the project was that I kept finding various functions I had forgotten about or didn't think I would need initially. Like for example a way to set the intractability of an object. This was pretty much always an easy fix but it's something I noticed happened a few times. I was pleased with how assembling the final game was easier than I thought because I had in a sense "front loaded" my work into the space class and its derived classes. Because each space handles its own interaction and modifies its state via accessing the player member function, putting the whole game together was actually simple. All I needed was to declare my rooms in main and set the directional pointers to the appropriate rooms for each room. Then after a quick menu that gave the options previously discussed was assembled, I started the program. Thinking I had no doubt forgotten something, I was surprised that the first time I put everything together and ran the program, I actually finished the game! This made me feel like I had made something truly modular as I realized how easy it would be in the future to just add another room into the game, set the proper pointers, and everything would work. Of course, this is the point, but it was really nice to experience all of the pieces falling into place much cleaner than I anticipated.

If I had more time I would probably work harder on trying to print things in a better way to make the game more aesthetically pleasing. As it was I noticed the output was pretty ugly at first due to lack of new lines where they would have been good to have. It's still not the best-looking thing in the world but it is better than it was, and that always gives me something to expand upon later!