Final Project Reflection

Byron Kooima - 932707491

CS/162

August 16, 2017

Prof. Luyao Zhang

Final Project Reflection

PLEASE SEE CHEATSHEET.txt IN PROJECT FILES TO QUICKLY MOVE THROUGH THE GAME!

Lab Description

The main focus of this program is to implement a one-player, text-based game where a player moves through Spaces to get Items and accomplish some defined goal. Aside from that, the basic game design was on the developer to design.

The implementation of the program would start with a Creature class and flow that down to each individual creature (Vampire, Barbarian, Blue Men, Medusa, and Harry Potter). The functional design needed to include a structure for the creatures to battle each other. Each of the creatures vary only in the values contained in the above table. The special attacks will override some of the attack or defense functions. Additionally, the entire combat simulation will have a test driver to test the "round robin" arena where all creatures fight each other at least once.

Requirements

- Make a Space base class (abstract)
- Space has at least 4 Space pointers:
    - Top
    - Right
    - Left
    - Bottom

  Each Space is linked to another with at least four pointers:
    - At least three different types of Spaces (derived)
    - Must have at least 6 Spaces within the Game
- The Game must have a theme:
    - A goal must be defined and achievable
    - Keep track of which Space the player is in
    - Must have a container for player Items

        - Must have a capacity limit
        - Items must be used as part of the solution to accomplish game's end goal
    - Must have a step limit
    - Player must interact with the Space structure
- Interface:
    - The goal of the Game must be declared at the beginning
    - No free form input
    - Provide user menu for each scenario
    - Visual or text based map

Class Design

Class Name	Space (base class)(hasA Creature, hasA userMenu)
Data Members	protected member variables: **Space* up;** **Space* right;** **Space* down;** **Space* left;** **userMenu subMenu;** **Creature* Villain = NULL;** **Creature* IamBat = NULL;** **bool specialItem;** **bool stunApplied = false;** **bool gasApplied = false;** **char menuChoice;** **std::string spaceName;** private member variables: public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions:

	Space(string, Creature*, Creature*) – Constructor: Set default values **virtual ~Space() –** Destructor **virtual char spaceMenu() = 0;** - Display the Space specific Menu **virtual void displaySpaceInfo() = 0 -** Display the Space specific Info **virtual void displaySpaceMap() = 0; -** Display the Space specific ASCII txt map **virtual void displayVillainInfo()-**Display the Villain Info assigned to the current Space **std::string getSpaceName() {return spaceName;} -** Generic function to return the Space's name **virtual void setAdjSpace(Space*, char) -** Default function for assigning the adjacent rooms to a passed-in Space. **virtual Space* moveSpace(char) -** Default function for moving from current Space to adjacent Space. **virtual void pause() -** Default function for waiting on user to push Enter **void printRound(Creature* pOffense, Creature* pDefense) -** Base level function to encompass all of the printed text and round status **void playRound(Creature* pOffense, Creature* pDefense, int round, bool player) -** Plays a single match between two creatures and prints the result of each round

Class Name	ArkhamAsylum (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **ArkhamAsylum(string, Creature, Creature)** – Constructor: Set default values

	~ArkhamAsylum() – Destructor **char spaceMenu() -** Menu input for the ArkhamAsylum Space **void displaySpaceInfo() -** Display the info screen for ArkhamAsylum **void displaySpaceMap() -** Display the ASCII txt map for ArkhamAsylum

Class Name	BatCave (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **BatCave(string, Creature, Creature)** – Constructor: Set default values **~BatCave()** – Destructor **char spaceMenu() -** Menu input for the BatCave Space **void displaySpaceInfo() -** Display the info screen for BatCave **void displaySpaceMap() -** Display the ASCII txt map for BatCave

Class Name	IvyLair (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions:

	(none) public member functions: **IvyLair(string, Creature, Creature)** - Constructor: Set default values **~IvyLair()** - Destructor **char spaceMenu() -** Menu input for the IvyLair Space **void displaySpaceInfo() -** Display the info screen for IvyLair **void displaySpaceMap() -** Display the ASCII txt map for IvyLair

Class Name	Warehouse (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Warehouse(string, Creature, Creature)** - Constructor: Set default values **~Warehouse()** - Destructor **char spaceMenu() -** Menu input for the Warehouse Space **void displaySpaceInfo() -** Display the info screen for Warehouse **void displaySpaceMap() -** Display the ASCII txt map for Warehouse

Class Name	WayneEnterprise (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)

Member Functions	private member functions: (none) protected member functions: (none) public member functions: **WayneEnterprise(string, Creature, Creature)** - Constructor: Set default values **~ArkhamAsylum()** - Destructor **char spaceMenu() -** Menu input for the WayneEnterprise Space **void displaySpaceInfo() -** Display the info screen for WayneEnterprise **void displaySpaceMap() -** Display the ASCII txt map for WayneEnterprise

Class Name	WayneManner (isA Space)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **WayneManner(string, Creature, Creature)** - Constructor: Set default values **~WayneManner()** - Destructor **char spaceMenu() -** Menu input for the WayneManner Space **void displaySpaceInfo() -** Display the info screen for WayneManner **void displaySpaceMap() -** Display the ASCII txt map for WayneManner

Class Name	Creature (base class) (hasA UtilityBelt)
Data Members	protected member variables: **int strength;** **int armor;** **int defPoints;** **std::string name;** **std::string creatureInfo;** **std::string specialAttack;** **int dieNumAttack;** **int dieNumDefense;** **int dieSidesAttack;** **int dieSidesDefense;** **UtilityBelt* uBelt;** private member variables: public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Creature(int, int)** – Constructor: Set default values **virtual ~Creature() –** Destructor **virtual void recover_strength = 0 –** Restore the creature to full strength **string get_name** – Returns the creature name **string get_creatureInfo** – Returns the Creature Info **int get_strength** – Returns the creature strength **int get_armor** – Returns the creature armor **int get_defPoints** – Returns the creatures defense **get_itemName** – Pass-thru function to get Item **string get_special** – Returns the creatures special attack **virtual int attack() –** Determine the attack damage from the appropriate dice **virtual int defense** – Returns the total damage taken by the creature. Updates strength based on total damage taken **virtual int pain(int attack, int defense**) – Determine the total damage inflicted **virtual void add_inventory(std::string, std::string, int, bool)-** Add Item to Creature's Utility Belt

	virtual void remove_inventory() - Remove Item from Creature's Utility Belt **void transfer_qty(std::string, Creature*)** - Transfer a single Item count from Villain Creature to Batman's Utility Belt **bool search_items(std::string iName)** - Search through Creature's Utility Belt for an Item **int die_roll** – Return the random die values based on each creatures specific die criteria **bool defeated** – Return a Boolean based on the current strength of the creature (<0 dead)

Class Name	Batman (isA Creature)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Batman(int, int)** - Constructor: Set default values **~Batman()** - Destructor **int attack**– Returns an int for a random attack value **int defense –** Returns an int for a random defense value **int pain(int,int)** – Takes a random roll of dice, adds it to the armor and then subtracts that from the attackers roll. **recover_strength** – Restores the Vampires to full strength **endScreen(bool)** – Prints the end credit screen

Class Name	PoisonIvy (isA Creature)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **PoisonIvy(int, int)** - Constructor: Set default values **~PoisonIvy()** - Destructor **int attack()** – Returns an int based on a random roll of dice **int pain(int,int)** – Takes a random roll of dice, adds it to the armor and then subtracts that from the attackers roll. **recover_strength** – Restores the Barbarian to full strength

Class Name	RasAlGhul (isA Creature)
Data Members	protected member variables: **(none)** private member variables: **ghulLives –** Counter to determine if Ra's has died public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **RasAlGhul(int,int)** - Constructor: Set default values **~RasAlGhul()** - Destructor **int attack** – Returns an int based on a random roll of dice

	int pain(int,int) – Takes a random roll of dice, adds it to the armor and then subtracts that from the attackers roll. **recover_strength** – Restores the BlueMen to full strength

Class Name	Riddler (isA Creature)
Data Members	protected member variables: **(none)** private member variables: **(none)** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Riddler(int, int)** - Constructor: Set default values **~Riddler()** - Destructor **int attack** – Returns an int based on a random roll of dice. **int pain(int,int)** – Takes a random roll of dice, adds it to the armor and then subtracts that from the attackers roll. **recover_strength** – Restores the Medusa to full strength

Class Name	Scarecrow (isA Creature)
Data Members	protected member variables: **(none)** private member variables: (none) public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions:

	Scarecrow(int,int) - Constructor: Set default values **~Scarecrow()** - Destructor **int attack** – Returns an int based on a random roll of dice **int pain(int,int)** – Takes a random roll of dice, adds it to the armor and then subtracts that from the attackers roll. **recover_strength** – Restores HarryPotter to full strength

Class Name	UtilityBelt (hasA Item)
Data Members	protected member variables: (none) private member variables: **Item* utilityBelt** **Item* aPtr** **int itemCount** **int arraySize** public member variables: (none)
Member Functions	private member functions: **int total_items()** – Calculate the total items for all of the Belt Items protected member functions: (none) public member functions: **UtilityBelt()** – Constructor: Set default values **~UtilityBelt() –** Destructor **void add_Items(string, string, qInput, special)** – Adds Item to Belt and checks to ensure the Item name does not exist in the array. **void remove_Items** – User selects Item from UtilityBelt array to delete. The array is re-sequenced if Item is removed. **void removeQty(string)** **void print()** – Prints the entire Utility Belt array for all of the Items. **void printItems()** – Prints just the Item name and quantity for displaying in the Game **Item returnItem(string)** – Returns an Item based on a passed-in search string

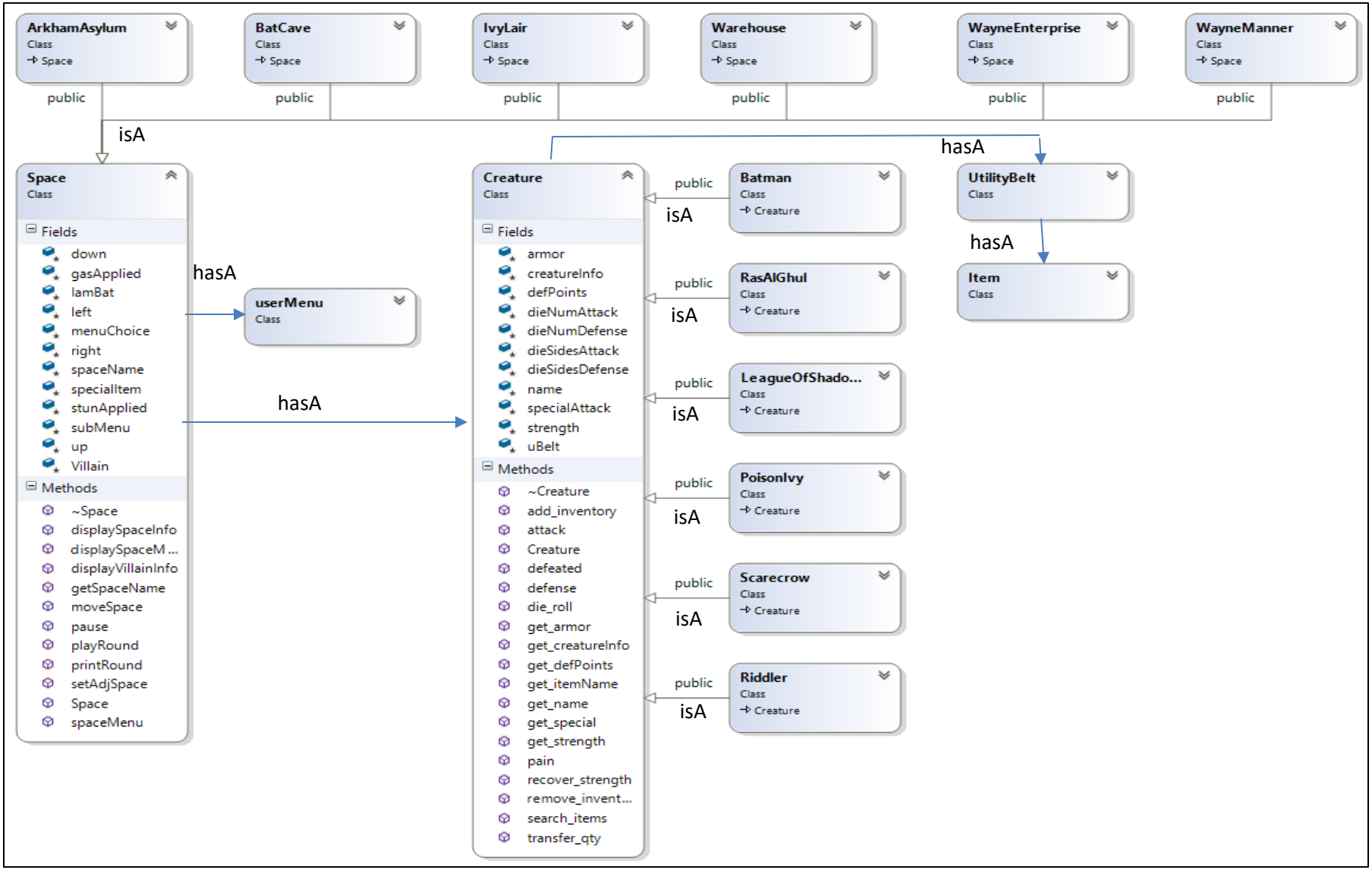
	bool searchItems(string) – Returns true/false if an Item exists in the Utility Belt

Class Name	Item
Data Members	protected member variables: **(none)** private member variables: **string itemName** **string itemDescription** **int quantityOfUnit** **bool special** public member variables: (none)
Member Functions	private member functions: (none) protected member functions: (none) public member functions: **Item()** - Constructor: Set default values **Item(string, string, int, bool) –** Constructor: user defined Item values **updateItem(const Item&)** – Updates an Item object with tempItem variable values **removeQty()** – Removes a single qty from the Item **print()** – Print the current Item object parameters **get_itemName() –** Returns the itemName (string) **get_itemDesc() –** Returns the itemDesc (string) **get_Quantity() –** Returns the quantityOfUnit (int) **get_special()-** Returns the Item's special status (bool) **bool Item::operator==(const Item& rhs) const –** Overloads the == operator to compare two Item names

Class Name	userMenu
Data Members	private member variables: **int selectedChoice** **vector<string> choice**
Member Functions	private member functions: (none) protected member functions:

	(none) public member functions: **void add_choice(string) –** Add menu options to vector **void printMenu() –** Print the menu options **int makeChoice() –** Returns int for the user selection

Class Interactions



Testing Plan/ Reflection

Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
Input too low	**Input < 0**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input not an integer	**Input = 0d**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input below range	**Input = -1**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
Input not a double	**Input = 34d**	**inputVerification -> SafeInput()**	**Prompt user to enter correct value**	**Prompt user to enter correct value**
User chooses option #3 to quit the program	**Input = 3**	**userMenu -> makeChoice()**	**Exit the program normally**	**Program exited normally**

I took a very iterative approach to this final project. I started out with classes I had already created for previous assignments. I laid out the design structure on a piece of paper and looked at all of the interactions between my classes and what functions I could reuse from them. I decided to use a combination of the new Space class and incorporate Creatures into it. Then I decided that each Creature would have a List (which I called Utility Belt). Finally, I used the doubly linked ideas to loop back Spaces to themselves. My final picture drawn on a piece of paper came very close to my final implementation.

The testing followed very closely with my design implementation. The additional aspects of my testing plan started out by testing a single Space. I built the BatCave object and tied it to four other Space objects (up,down,left,right). I then placed a single Creature into the Batcave. Once I tested that Batman could move from one Space to another, I implemented Batman's Utility Belt.

I then created six separate rooms, each with their own unique characteristics. I also added a unique Creature/Villain to each of those rooms. Then I went back to my drawing board and started coming up with the tool list that Batman would need to traverse the difficult landscape.

After I decided on the best tools for each Villain, I decided to add the appropriate menu for each of the Spaces. I tested to make sure a simple case within each room would provide the expected result. I then put checks in place to make sure moving from one room to another had some sort of requirement. Those interactions started introducing some of my first bugs. I did not have a great plan on how to make sure Batman would only interact with the menu option one time and only receive one of any "required" item.

So, back to the drawing board I went. I tried to trace the up-down relationship on how Batman could get an Item into his Belt. I ultimately decided that each Villains' Belt should also be loaded with Items. This meant I had to create some way for Batman to "take" an Item from a Villain's Belt. In the end, my solution was somewhat clunky and if I had more time, I would have tried to implement it differently. I ended up having to create many additional functions within the Base Creature class so I could reach all the way down into the Items list. I know there was a better way to access the data members from Items but I couldn't figure out how to go down from Space -> Creature -> UtilityBelt -> Item. I started having success with static_cast but ran out of time to fully implement it.

From that point on, the complexity of my idea started to grow and grow. I opted to continue testing pieces at a time and each time I incorporated another Object, I would test just that functionality. While this worked very well in the beginning, I did have to draw up some new interactions between my classes.

For starters, when I decided to incorporate the Combat Game from Project 3, I had to figure out what Items would increase attack strength of the Creature and what Items would help the Creature's defense. Then it was a bunch of rounds of play between two Creatures to see how the Utility Belt Items would increase or decrease Batman's chances of success. My logic

diagram for this interaction started to get a little over the top. Mainly because I had to make sure that Batman either had a way to revive himself or to make sure each combat wouldn't take too much of his life.

Ultimately, I opted to very heavily weight the Items Batman was carrying. This meant that if Batman decided to take on any Creature/Villain before acquiring the required gear, he would pay the price. For the early Villains, Batman had enough strength and armor to defeat them without taking much damage. But the next Villain in the chain could cause substantial damage if he didn't acquire the gear from the first Villain (and so on and so forth). This allowed the user to try and take on the League of Shadows (second to last Villain) and succeed, but without some of the other items gathered along the way, Ra's Al Ghul would kill Batman fairly quickly.

The combat scenarios were where I spent a significant amount of time so I could get the gameplay just right. I wanted the user to be able to take on any Villain at any time, but their chances of success were severely diminished if they don't follow the prescribed path. There are hints along the way and I made sure to check Batman's Utility Belt before every battle to make sure the user really wants to take on a Villain without the proper gear.

Overall, I am pretty pleased with the result. I spent way more time than was probably necessary creating nice ASCII images and maps. If I had to do it again, I would have spent a little less time on the artwork and more time on cleaning up my code. Because I used so many Classes from previous assignments, I didn't initially assess the value of certain functions. It turned out that many of my functions could have been removed all together. I did clean it up as much as I could in the time I had left, but I would have done some additional refactoring if I had another week or so.