Final Project Design + Reflection
Tay, Serena
Student #: 933588509

Document (Design + Reflection)

You need to create your program design **BEFORE** you start coding. Your design needs to be included in your document.

Simply stated, **program design** is identifying the problem to be solved, modularizing and describing the structure of program, and making a test table to that fulfills the requirements of the program.

For test table, please include test plan, expected output, and actual output. The test plan should be robust. The expected output is based on the requirement. If your actual output deviates from your expected output, that usually means the program behavior is deviating from the requirements. Following is a Examples Test Plan.pdf

In your **reflection**, you should explain what design changes you made, how you solved some problems you encountered, and what you learned for this project. The TA will grade on how well your implementation matches your design.

Design

- Create a master parent space class with the following member variables and functions to be inherited to by all derived classes.
- Six derived classes Shire, Bruinen, Rivendell, Rohan, Gondor and Mordor are derived from the Space class.

Space Class

Contain the following member variables to store Space data as outlined in the project requirements:

```
protected:
  Space *top;
  Space *down;
  Space *left;
  Space *right;
  string spaceName;
  bool  playerPresent; //to detect if player is present

public:
  Space();
```

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

```
virtual bool event() = 0; //virtual function for an event that occur in every derived class
string getSpaceName();

void setPointer (string, Space*);
```

event () - With a pure virtual function event that every derived space will override with a unique event, it meets the definition of an abstract class.

Data validation/integrity consideration for Space class:

- Nothing as this is the parent class that does not get any user inputs

Derived Space Classes

- All contain the same member variables and functions
- The pure virtual function Event is overridden in every space classes where each spaces contain a unique quiz/combat challenge that a player needs to pass in order to advance to the next step

Menu Class

Contain the following member variables:

- countPlay – This is used to determine if the "new run" menu should be used or "play again" menu should be used per requirements
- menuChoice – used to store choice entered by user

Contain the following member functions:

- Menu () – default constructor to initialize member variables used
- displayMenu() – to display menu (contained 2 menus, one for the new game, one for the 'play again' scenario once the first Game run occurred
- prompt(string, int, int) – a generic function that can be used to prompt user for data inputs. Contain 3 parameters, one string to tell user which member variable the program is asking for, one integer for the minimum allowable input and one for the maximum allowable input. **Please note that this function was discarded and not used**

Data validation/integrity consideration for Menu class:

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

- make sure user gets an error message when entering an invalid choice
- create a max and min for a range of allowable input (create an error message when user entered an invalid number outside the range)
- make sure non-ints are guarded from creating errors in the program

Game class
Contain the following member variables:

```
private:
  int stepCount,  //keep track of limit of steps a player can take
      playChoice; // user inputs choice
  //pointers to all 6 spaces and current space and player
  Space *shireSpace;
  Space *bruinenSpace;
  Space *rivendellSpace;
  Space *rohanSpace;
  Space *gondorSpace;
  Space *mordorSpace;
  Space *currentSpace;
  Player *player;
```

See comments for explanation on member variables.

Contain the following member functions:

```
Game(); //constructor
  // ~Game();
  void runGame(); //run game in sequence
  void createCharacter(int); //dynamically create player
  bool runAdventure(); //run adventure in 6 spaces
  void createSpaces(); //dynamically create spaces
  void deallocatePointers(); //release memory and set pointers to null
  void updatePouch(int); //add prize into pouch
```

- Game() – default constructor used to initialize all the member variables listed above
- createSpace – To dynamically create Spaces

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

- runGame() – used to run the Game and calling the menu and Space classes in sequential order (display menu, ask user for input, run program, and repeat until user want to quit)
- runAdventure – initiate the sequence of game in six spaces and to stop when step count reaches 10 when the game hasn't finished
- deallocatePointers – to deallocate pointers to avoid memory leaks

Data validation/integrity consideration for Game class:

- Use menu functions to ensure user inputs are valid

Player Class

```
protected:
string playerPouch [5]; //container for the player, to carry "items".
string playerName;
int pouchCount; //keep track of pouch limit


public:
  Player();
  void insertPlayerPouch (string, int);
  void printPouch ();
  void setPlayerName (string);
  string getPlayerName ();
```

Designed to ensure the player carries a "pouch" of an array to store tokens

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

Testing plan and results

Menu class

- displayMenu() – to display menu (contained 2 menus, one for the new game, one for the 'play again' scenario once the first Game run occurred

Test Case	Input Values	Expected Outcomes	Observed Outcome
Display 1st menu when program initialize	Menu counter	Display first menu	Success – Achieved expected outcomes
Display "play again menu" after first run	Menu counter	Display play again menu	Success – Achieved expected outcomes

- prompt(choice) – a generic function that can be used to prompt user for choice of which Spaces they want to use

Test Case	Input Values	Expected Outcomes	Observed Outcome
Program select correct Space based on user choice	User input 1,2,3,4 or 5	Display correct Space names when combat starts based on user inputs	Success – Achieved expected outcomes for all inputs prompted for
Display error messages if input does not fall between the min and max set	Game class calls	Display appropriate error message when user input is outside of valid range set and reprompt user until valid input is provided for valid Space choices (1 to 5)	Success – Achieved expected error message outcomes for all invalid inputs and reprompt user for valid input

Game class

- Game() – default constructor used to initialize all the member variables listed above

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure constructor initialize to default values	N/A – set in program	Member variables initialize to the values stated	Member variables initialized to the values stated

- runGame() – used to run the Game and calling the menu and Animal class in sequential order (display menu, ask user for input, run program, ask user what they want and repeat until user want to quit)

Test Case	Input Values	Expected Outcomes	Observed Outcome

Ensure function is run the function calls in sequence and keep looping until user decides to quit	N/A – set in program	1. display menu 2. run program 3. link spaces in sequence (shire, bruinen, Rivendell, rohan, Gondor and Mordor 4. currentSpace pointer points correctly to space and go through the linked list of spaces 5. ask user if they want to play again with menu display 6. Repeat loop until user want to quit	Success – Achieved expected error message outcomes for all invalid inputs and re-prompt user for valid input

- createSpace – To dynamically create Spaces based on user choice

Test Case	Input Values	Expected Outcomes	Observed Outcome
Dynamically create all 6 spaces And link them in a linear sequence	User input	Function dynamically create Spaces as prompted. Linked in the following sequence - Shire - Bruinen - Rivendell - Rohan - Gondor - Mordor	Success – expected outcome achieved

- runAdventure – initiate the adventure in sequence

Test Case	Input Values	Expected Outcomes	Observed Outcome
Ensure rounds are executed in sequential order (see above)	Space class inputs	Rounds are executed in order. Player 1 attack, then player 2, then repeat until a player dies	Success – expected outcome achieved
Events () - Shire - Bruinen - Rivendell	User's inputs	Ensure the program validates user's input accurately and prompt user to try again if it doesn't meet the minimum requirements to pass	Success – expected outcome achieved

- Rohan - Gondor - Mordor			
Update Pouch	Game file input	Ensure that the player collects all the prize in order to win the game	Success – expected outcome achieved

- deallocatePointers – to deallocate pointers to avoid memory leaks

Test Case	Input Values	Expected Outcomes	Observed Outcome
Memory leak	N/A	Run program with Valgrind to ensure no memory leak	Success – expected outcome achieved

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

Reflection

Inheritance

- virtual functions are used heavily in this program which allow Spaces to be created with
- setting up the member variables and functions in the Space class carefully enable all derived Spaces to inherited the functions and reduce the time and effort significantly to re-create these member variables in each of the derived class

Abstract Class

- Choosing which function to be the virtual function was easy for this project as given it should be a game of different challenges, I made the event () pure virtual

Linked List

- After last week's lab, this is the first time I used linked list in my program and realized how linked list can be convenience in going through a linear list

Memory Leaks

- I encountered a lot of memory leaks in this project given extensive use of pointers in the linked lists. I started by going through all the dynamically created objects and ensure they are properly deleted and points to NULL
- I discovered that I initially created a struct that was not used in the program which was triggering the memory leaks, once deleted, the issue was resolved

Bugs

- There was a weird bug in my last space "Mordor" where it seems that the pointer is pulling the name of the object in a pouch rather than the player's name when required
- This was resolved after discovering the array size was one size smaller than required to fill all 6 items

In hindsight, I wished I learned more about vectors, STL contains and leverage their functionalities to make the game more interactive and powerful.

Design

- It was hard to keep track of the results and ensuring the damage is applied accurately in the combat events in Mordor and Gondor. Embedded a print function helped made the program easier to test
- Initially I had issues couting the content of the pouch as it keeps couting the last prize obtained by the number of rounds the player has advanced. I should realised that it was because I set the array count to the last count and not [i] in the for loop.
- I started by keeping the design simple and not making anything complex and sure it is running before adding more complexity to the design
- For every space the player succeeds in the event/challenge, the player obtains a prize that gets inputted in an array calls player's pouch.
- Ensure each spaces have an event that requires user's input to ensure they are involved in the reaction/result of the games

Final Project Design + Reflection
Tay, Serena
Student #: 933588509

Testing Phase

- Testing strategy included the following as outlined in the testing table above
    - Out of bounds, invalid user inputs to prompt error message
    - Accuracy of Character chosen based on their input
    - Accuracy of damage calculation in Gondor and Mordor events
    - Program handling events/situation properly
    - Program inputs users input for choices to be made accurately
    - Ensure memory leaks are resolved using Valgrind