ChamberCrawler3000 (CC3K) Design

Introduction

The design document will begin by discussion the changes between the plan of attack and the final design of the project, stating what has changed or not changed, and what features have been added/removed.

Next, the document will give an overview of the game on a high level, and go then go into detail to each aspect of the game (essentially the same as provided in the plan of attack).

Subsequently, the details of the implementation of each feature and aspect of the game will be discussed, including any design pattern usage and software engineering technique. This will be followed by the discussion of the bonus content and how they were implemented

In addition, the answers to the questions in the assignment specification will be provided again, including any change to them.

Finally, a conclusion statement will be provided, answering the final questions at the end of the project guidelines document.

Overview of the change between plan of attack and final design

The overall design, on a high level, has not changed for the game that satisfies only the basic specifications. However, multiple downloadable content has been added to the game as bonuses, including random map generation, using the ncurses library to

implement keypress control of the player, an inventory to store potions, and the ability for the player to buy potions from merchants.

Overview of the game

CC3k will contain a main game grid that presents the dungeon layout, including walls, floor tiles, doorways, passages, and empty space.

The game consists of 5 main levels, represented by floors. Each level has the same default layout.

There will be NPCs (non-player characters) that are spawned in the dungeon, including different types of monsters (human, dwarf, elf, orc, merchant, halfling, dragon) and a stairway to the next level.

The player will take on a role that is a hero slaying the monsters, collecting the treasures, and reaching the last floor's stairway. Upon reaching the last floor's stairway, the game is won and the player's final score will be based on the amount of gold he/she has.

The player can choose to become predefined race: drow, vampire, troll, goblin, each with their own abilities. The player can find potions, which are items that can modify the player's health, attack, or defence, or gold on the dungeon floor as he/she progresses through the dungeon.

The dungeon grid

The dungeon is an object of the singleton DungeonGrid class, and contains two-dimensional grid of game objects. The plan is to use a pointer to an object of the

GameObject class for each slot on the grid so that any game object can be placed onto the game grid.

First, the dungeon grid is responsible for generating the floor layout, the game object locations, that is, the spawning locations for the player, stairway, monsters, potions, and gold.

Next, the dungeon grid will, upon the command of the user, take action with the player, be it to move to a different location on the grid, use a potion at a location on the grid, or attack a monster at a location on the grid. The dungeon grid will use

The precedent actions will continue until the player is dead, the player decides to restart, the player decides to quit the game, or the stairway in the fifth floor is reached by the player. Once the player is dead or if the player decides to restart the game, the user will be able to select another race. But if the user quits the game or if the stairway in the fifth floor is reached by the player, the game will end, and the latter triggers the display of the player's score.

The player and the monsters

The player can becoming one of the following races: drow, vampire, troll, goblin, or he/she can remain a shade. The monsters include human, dwarf, elf, orc, merchant, halfling, dragon.

The player will have interactions with potions and gold on the floor, as well as monsters that are nearby. The interaction logic is handled by the dungeon grid, but the details, such as how the damage calculations are determined, are calculated within the player and monster classes.

The potions and gold

There are six types of potions: restore health, boost attack, boost defence, poison health, wound attack, and wound defence. There are four types of gold: small hoard, normal hoard, merchant hoard, dragon hoard.

The interaction logic between the player and potions/gold will be handled by the dungeon grid. But the effect of each type of potion/gold is within its own class.

Implementation details for the basic game

The dungeon grid

The dungeon grid class employs the Singleton design pattern. There will only be one object of this class existing per program execution.

The dungeon grid provides numerous methods for interaction, mainly to provide interactions between the game objects on the grid.

At the start of each game session, the dungeon grid will be responsible to load the map from a text file. The text file will be of two types: one is the default map provided in the assignment specification, whereas the other is a file provided in the command line argument that provides the layout of each of the five floors in the game.
This loading of the map will spawn all the walls, floor tiles, passages, and doorways on this floor.

After loading the map, the dungeon grid will determine the bounds of each chamber (in the case of using the default map), which is useful during the spawn stage of all the game objects.

Upon the determination of the chamber locations, the dungeon grid will spawn the game objects in the following sequence: player, stairs, potions, gold, enemies. There will only be one player and one stairs. There are 10 potions and 10 golds, and there will be 20 enemies.

These objects will all be spawned in a similar fashion: first a chamber is selected randomly, then a row in this chamber will be selected randomly, and finally a column in this selected row in this chamber will be selected randomly, and then the object is placed at this spot.

In the case for the stairs, the chamber for its spawn is continuously selected until it is not the same chamber as the player.

In the case of the potions and gold, 10 of each will be spawned this way in a for loop.

One particular note about the spawning of the gold is that if a dragon hoard is to be spawned, a dragon will spawn in its one block radius and this dragon will count for one of the 20 enemies to spawn on this floor level.

Finally, in the case of spawning enemies, the for loop will be of 20 - number of dragons count iterations.

When the game objects are spawned, the game starts and the dungeon grid will be responsible to handle the events happening the game.

At each turn for the player, the player may choose to attack a location, using a potion from a location, or move to a location. These three events are all handled by the dungeon rid.

In each of the three events, the dungeon grid will assess the target location of the action, determine the validity of the action, and return the appropriate message if the action was not valid. If the action was valid, then it will carry out the action, whether it's the player moving to the location, the player using a potion from the location, or the player attacking the object at the location (the details of the events are discussed further in the discussion of the implementation of the player, enemy and potion). If the player

successfully moved to the target location, and there is a gold at that location, then the player will pick up the gold and the player's gold count will increase according to the type of the gold. However, in the case of the dragon hoard, the player won't be able to pick up the gold unless the dragon in its one block radius is dead.

After the player's turn is taken, the enemies will move. To prevent duplicate movements of the enemies, an array of enemy pointers stores all the enemies once is used to move all the enemies. The enemies will move according to their type by the use of a virtual method to move. For example, a human or a dwarf will attack the player if the player is within their one block radius or move to a random location within their one block radius if no player is present in their one block radius, whereas a dragon will not move but only attack the player if the player is within one block radius of it or the hoard it's guarding.

When the stair is reached by the player, the player will go to the next floor of the dungeon and the dungeon grid will clear all the game objects on the grid and spawn new objects accordingly and the cycle starts again.
The game ends when the player reaches the stairs at the fifth level, the player dies, or the player restarts. All of these events trigger the clean up of the grid in addition to the player, but only the first case will have the player's score printed out, which is the player's gold accumulation or, in the case of the shade race, 1.5 times the player's gold accumulation.

At program termination, which occurs when the player chooses to quit the game, the dungeon grid will be destroyed and all of its grid and chambers will be destroyed, deallocating all the memories they have allocated at the beginning of the program.

The game object

The game object is a super class of all the objects in the game, be it a floor tile or wall, a player or stairs, or an enemy.

From the game object, several major sub-superclasses are inherited: player, enemy, potion, gold. Only the cells of each floor directly inherits from game objects because they do not require common actions to be used in interactions with other game objects.

The player

The player is a superclass that has all of the races inherited from it. The player provide virtual methods for attacking an enemy, being attacked by an enemy, picking up gold, and HP regeneration.

Each race of the player will use the default implementation of the virtual methods unless they have special abilities that require their own overridden version of the methods. At runtime, polymorphism will determine which method is used given the race of the player. In the basic game, the shade and the drow are the only two races that don't override any of the virtual methods in the player class because their special abilities don't require them to.
The vampire and the goblin override the attack method to gain 5 HP upon each attack and gain 5 gold upon each enemy death by the player's attack, respectively. However, the vampire will lose 5 HP if the target of the attack is a dwarf.
The troll overrides the HP regeneration method so that the player regenerates 5 HP at the beginning of every turn.

The potion

The potion is a superclass that has all of the different types of potions inherited from it. The potion provides a virtual method for the player to use the potion.

Each type of potion will override the virtual method for player using the potion to implement their own abilities.
The restore health and poison health potions will use the player's adjust HP method to increase and decrease the HP of the player by 10, respectively.
The boost attack and wound attack potions will use the player's adjust attack method to increase and decrease the attack of the player by 5, respectively.
The boost defence and wound defence potions will use the player's adjust defence method to increase and decrease the defence of the player by 5, respectively.

The gold

The gold is a superclass that has all of the different types of gold inherited from it. Each type of gold will have their own values.

The small hoard has a value of 1, the normal hoard has a value of 2, the merchant hoard has a value of 4, and the dragon hoard has a value of 6.

Small hoards and normal hoards will be spawned on the floor during the objects spawning stage of the game.
The merchant will only be dropped when a merchant is slain by the player.
The dragon hoard will spawn on the floor during the objects spawning stage but won't be able to be picked up by the player unless the dragon guarding it is dead.

Enemies

Enemy is the superclass of all different types of enemies. Each enemy has different stats for starting HP, attack, and defence. Some enemies have special abilities.

The enemy class provides virtual methods for checking if it's dead, moving on the dungeon grid, attacking the player, and being attacked by the player.

Each enemy will drop either 1 or 2 gold upon death. This gold will be added immediate to the player's gold accumulation. In the case of coding, a random generation of 1 or 2 gold is directly added to the player's gold count instead of creating actual objects of small or normal hoards and then add them to the player's gold count.

The human, unlike other enemies, will have 4 gold (equivalent to 2 normal hoard's amount) added to the player's gold count upon death.
The elf overrides the attack method of enemy and attacks twice unless the player is using the drow race.
The halfling enemy overrides the method of being attacked by the player and dodges 50% of the time when attacked by the player.
The merchant is neutral to the player unless it's attacked. This is implemented by a static variable of the merchant class to determine hostility. Upon the attack of one merchant, the hostility variable will change value by a static method and all merchant will attack the player given the opportunity. The merchant also overrides the move method of enemy and will attack the player if the hostility variable is true. One note about the hostility variable is that it's set back to the value by a static method such that all merchants are neutral to the player when a new game session starts. The merchant also drops a merchant hoard upon death and can be picked up by the player.
The dragon overrides the method to check the enemy death and to move. Each dragon object contains a pointer to the dragon hoard object that the dragon is protecting. Upon death, the dragon uses this pointer to set the dragon hoard free and the player will be able to pick up the dragon hoard. This is done by a boolean value in each dragon hoard to determine whether it's free or not. The dragon also cannot move and will only attack when the player is within one block radius of either its hoard or itself.

Bonus content

Random map generation

The first of the DLC is the random map generation. Instead of the basic floor plan provided in the assignment, if the user uses the DLC version of the game, when no floor plan is provided, random map will be generated consisting of 5 chambers of randomized width and height.

The random map generation implementation is as follows: the 5 chambers are looped through and at each chamber, the number of rows is first randomly generated, and then the number of columns is randomly generated, then the passage and door locations will be determined given the width and height of the chamber. Each chamber will be connected through a given, minimally defined set of passages.

Ncurses keypress control

If the player desires to play the DLC version of the game, then the player will be able to control the player using a previous defined set of keys to denote the directions to which the player will move, attack, use potion, pick up potion, or buy potion from a merchant.

Due to the fact that ncurses will have its own window to display information, addstr() is used instead of cout for proper formatting.

The inventory of potions

In the DLC version of the game, the player has a display of his/her inventory, which is a 10 slot container for potions. If the player desires, he/she can pick up a potion from the floor instead of using it and put it into the container. Then he/she can use a number from

1 to 0 (on the keyboard) to use the potion at that particular spot in the inventory. Upon the use of a potion in the inventory, the rest of the potion at slots after the used potion will be pushed up a slot.

The implementation of the inventory is done by adding an array of potion pointers in the player class and a method to add a potion into the array. Upon deletion of the player, the destructor of the player will clean up the inventory if needed.

Merchant sales

In the DLC version of the game, the merchants have the ability to sell the player potions.

The player can press a defined button to buy from the merchant at a location within the player's one block radius. A new screen will appear asking the player what type of potion he/she wishes to buy. The transaction will end if the player desires not to buy anything anymore. The transaction won't be successful if the player's inventory is full or if the player has insufficient gold.

There is an addition method in the dungeon grid that handles the buying of potions by the player from the merchants.

If merchants are hostile to players, they will refuse to sell the player any potion. This is done by checking the hostility variable in the merchant class when the player wishes to buy from merchants.

Questions on the project specification

The changes from the plan of attack are highlighted below

1. There is a super class called Player, and the races will be sub classes that inherit from Player, with shade being an actual Player object. This solution will make adding new classes not very difficult because the class system should be a decorator system for the Player super class
2. There is a super class called Enemy, and the different enemies types should be sub classes of Enemy. This is the similar to that of the player, because there are many fields (e.g. hp, atk, def) that are shared among all enemies.
3. The enemy abilities and the player race abilities is handled by the interaction logic because they are simple combat abilities that do not quire their own classes. However, if more complicated skills/abilities are to be added, there might be a need for a Skill/Ability super class.
4. From plan of attack: “A visitor design pattern is used to model the potions' effects, since this way the program will know which potion are being used on what character”; Change: A visitor design pattern was actually not required as polymorphism of the potion and the player was enough to determine how the potion is used on the player. For example, a boost attack potion will call the adjust attack function of the player. Also, a potion effect multiplier was also used to implement the increased potion effect for drow.
5. Both potions and gold are spawned in similar fashion. First a chamber in the floor is randomly selected, then a row of that chamber is randomly selected, and then finally a column of the selected row of that chamber is randomly selected and the potion/gold will be placed there. This way, the only difference between the generation of gold and potion will be their probabilities.

Final questions

1. This project has taught us that developing software in teams requires vigorous planning, hence the preparation of the attack plan in due date 1, and the UML creation for structuring the program. It also taught us that communication is a very

important aspect in developing software in teams. If we didn't communicate properly, then the implementation of the game might have been difficult and we would have had to spend more time debugging because our parts don't match in implementation. Also, in developing software in teams, sometimes it's difficult to come to a single solution to a problem as each developer has his/her own mind in what the solution should be.

2. If we were to start over, then we would plan out every detail of the game more thoroughly, so that some of the implementation difficulty that we've encountered could be avoided because we didn't consider this or that detailed of the game.