CS4303 AI Practical: RogueLike 160026526 21 March 2021

• Overview:
The aim of the practical was to implement a “Roguelike” in Processing, using the concepts from the AI and Procedural Generation portions of the module and have though on game design, implementation and tuning.

• Design:
Basic dungeon procedural generation
The design for each level of the dungeon would be based of an array of arrays with 1’s and 0’s to represent the spaces that the players, mobs and items can spawn and move in based off a mapSize. A room class is made to construct rooms in each floor and place an array of an array to represent the tiles which will be placed into the dungeon. A dungeon class is then created which will be used to decide where to place the rooms and tunnels connecting all the rooms. PlaceRooms is then used to place rooms in the tiles of the level, it first randomizes the total number of rooms that can be placed based on a minimum total of rooms and a maximum total room based off two ratios and the total rooms that could be placed in the level. For the sake of consistency, both min and max ratios are set the same but if necessary, could be tweaked to decrease or increase the number of rooms in each level. A for loop is loop around the chosen totalRooms that was generated pseudo-randomly which generates room cells to be placed in empty cells in the dungeon. This is stored in the list of all the rooms. Random offsets after each x and y position for a room is used to make the rooms less symmetrical to one another. The tunnels connecting each room are placed using placeTunnels. The method connects each room to its previous and next rooms. This is done by finding the center of both rooms and the x and y distances between rooms. Tunnels are added in both x and y directions at random and in random lengths until the center of both rooms are reached, which connects the two tunnels together. The map of the level can then be built using the width and height of desired level by making all the tiles non transversable, placing the rooms and then the tunnels to connect the rooms. The dungeon builder is created in the main class and a newMap is built using the dungeon builder. The level is drawn in by iterating over the map and drawing rectangles for the movable areas as seen in examples from lectures. A background was placed for the level to show the untraversable areas. The dungeon generation algorithm was adapted from http://www.codingcookies.com/2012/08/07/procedurally-generating-dungeons/.

Player and mobs position design
The player is placed in the first room in the rooms arraylist on a random tile, a mob is spawned in every other room on random tiles.

Player movement design
In the beginning, the player was constructed with x and y values for its position and, similar to lectures, was drawn as a circle with a dot to show direction facing. The integration for the player was implemented using the kinematic algorithm from lectures, using the arrow keys to control the velocity change of the player in the corresponding direction. To allow for collision detection with walls, a wall class is created, and an array of walls are used to store all walls. The array is then iterated over in draw to check whenever the player collides with a wall in the level. Boolean switches are used to control the player movement.

Mob movement design
In the mob class, the mobs movement also uses the kinematic algorithm from lectures. Mobs move in 8 different directions, the regular cardinal directions and the diagonals. At first mobs were able to move outside the rooms they were placed in but was removed and made each mob unable to leave their room as there were occasions mobs would congregate by accident in a single room and became unable to leave. The collision detection for the mobs and the walls designed so that once the

mob collides with its room walls, it will move in 1 of the 8 directions. Wanting the mob to chase the player, in the integration for mobs an if statement was used to check if a player is in a radius 5 times the mob size and if so the mob will accelerate towards the player using the steering algorithm.

Attributes for characters

Player attributes were added including health, player level, experience, etc. The rules are then decided on how the game would play out here. The player STR stat will correlate to the damage the melee attacks do and the DEX stat will correlate to damage from ranged weapons. To differentiate them, melee attacks will always have a higher base damage than ranged weapons but ranged weapons have a 30% chance of dealing 1.5 times the damage. A SPD stat will show difference in the players speed when moving which also increases when levelling up. As for mobs, 4 different types of mobs were decided on which all have different health, attack, expYield, sizes and speeds. Slimes being the smallest and fastest while Giant Crabs being the largest and slowest but with a larger player detection area to steer towards. In the mob class, the mobs attacks are decided by die throws from 0 to 9. Mobs have different chances to hit or miss based on these odds. An example is the slime which has a 50% chance to do nothing. For the two stronger mobs, the Giant Crab and the Vampire, there are a few additional rules. For the crab, if the player's DEX is more than 8, the crabs attacks will always deal no damage. This is used to emulate nimbleness as the Giant Crabs are slow wherelse vampire have a chance to deal twice the damage and if the player is under a health threshold, the vampire has a higher chance of dealing twice the damage. These mobs have a small chance to be place in the initial room but have a higher spawn chance in further levels allowing for the player to learn early to not try to engage them in the small chance the player comes across them. Attack descriptions also correlate to the mob attacks. In the player class, there is a method to getEXP which will be used when player defeat mobs and when the exp threshold is filled, the player levels up and stats increase by a base amount.

Status page

A status page was desired to view the players stats. This was done using Boolean switches to pause the overworld phase and turn on the stats page in draw using the 's' key press. All the players stats are displayed in the status page

Combat system

Combat occurs when a mob touches the player, although the mobs and the player are both drawn as circles, a square 'hitbox' around them is used for the collision detection in the draw method. When collided, all player movement is stopped and the fightingMob is set to the mob collided with and the battle and encounterMessage Booleans are set to true while the overworld Boolean is set to false to pause the gamestate. When battling, depending on the mob type, a different background is used. The players level, experience and health are in constant display for the player to see. The players first attack will be when encounterMessage is true and then the rest when playerTurn is true. The player can attack with melee and ranged weapons with keypress 'a' and 'b' or run with 'r'. After the keypress, the mobTurnA/B/R is switched on to display the damage dealt or if the flee was unsuccessful. To continue, the player will use the 'enter' key to continue the game back to the players turn and the mob will have attacked based on a diceroll from 0 to 9. This is continued until either the player or the mobs health reaches <= 0. When the mob dies, the player gains its expYield and can proceed by clicking 'enter'. The player has a 50% chance of running away. When first implementing running, the mob would immediately be encountered again. To stop this from happening, a 3 second and 2 second timer was used to, only move mobs after 3 seconds of running the overworld draw and the collision detection with mobs will only activate after 2 seconds to allow for a grace period when running is successful. When mobs are defeated, they are set to a dummy x and y position as to not be encounterable again. When the player is defeated however, the player can choose to restart the game by pressing the 'enter' key. Text display in a rainbow gradient when mobs is defeated to show player winning and text is displayed in red when player dies.

Item class

An item class is used to create items which have ids, names, descriptions, types and function. At the moment 3 types of items exists which are of “Melee”, “Ranged” and “Potion” types. The function is an int which correlates to the type of item. When melee or ranged weapons are equipped, the func value is added to the STR and DEX stat accordingly. Wherelse, when an item is of type potion, the player will restore health equal to func when using the item if the health is not already full. These are done in the useEquipment method in player. An arraylist is used to store all items in the inventory and the player has a currMelee and currRange items. To allow for the usage/equip of these items, the Status page is extended to also display all the items in the player’s inventory. When ‘e’ is pressed while in the stats menu, a Boolean equip is set to true to allow players to select the item using ‘<’ and ‘>’ to use or equip. The key ‘e’ can be pressed again to equip or use the item. It is displayed clearly which item is being selected and what key to be pressed in the stat menu along with an item description. The player is always equipped with ‘hands’ and ‘rock’ initially.

Adding items to the dungeon

Initially, three items are generated in the same room as the player in random positions in the room. These includes a melee, ranged and potion items. The weapons are equipable at the start. In each other room, there is a 30% chance to generate a regular potion and a 5% chance to generate a large potion. Potions are only usable outside of combat.

Goal of the game

It was decided that the goal of the game was to escape from 5 levels. An exit class is used to place an exit on each level and collision detection of the player with the exit will progress the game by incrementing a level int in the main class. Treasure items are also added, one per level and are disguised with the same icon with regular items. This incentivises players to collect every item in each level to ensure they also get the treasure. The scoring system involves calculating the total number of treasures along with the players level to determine the score. Therefore, players should also be actively killing mobs. When exits the fifth level, a win screen is displayed, which shows the player’s score and the number of mobs the player killed. The player can then choose to play again by pressing the ‘enter’ key.

Choosing which mobs to spawn

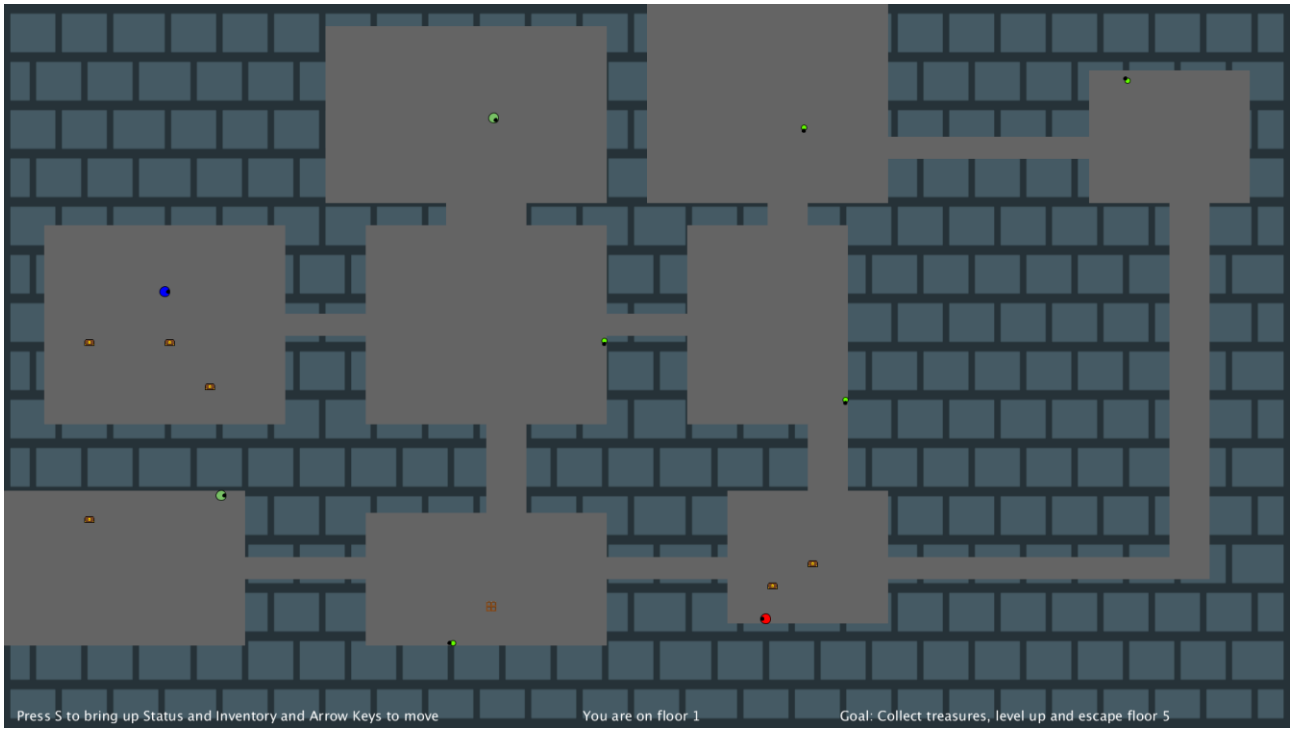
Each level contains a curated spawn rate for each of the four mobs. This is to ensure that there is a difficulty increase in each level. The spawn rate for the fourth level in particular includes a higher chance for giant crabs to spawn. Giant Crabs will be easily defeated if the player has been killing a large number of mobs in the early levels, so this would be a bonus room and if they haven’t, the room would provide a challenge. The last level generates more vampires so should be the most challenging room. There is a small chance for the hardest enemy, the vampire, to spawn at the beginning of the game and if one does and the player encounters it, it will most likely end in a loss and players will learn to avoid it on future runs.

Weapon spawn on higher levels

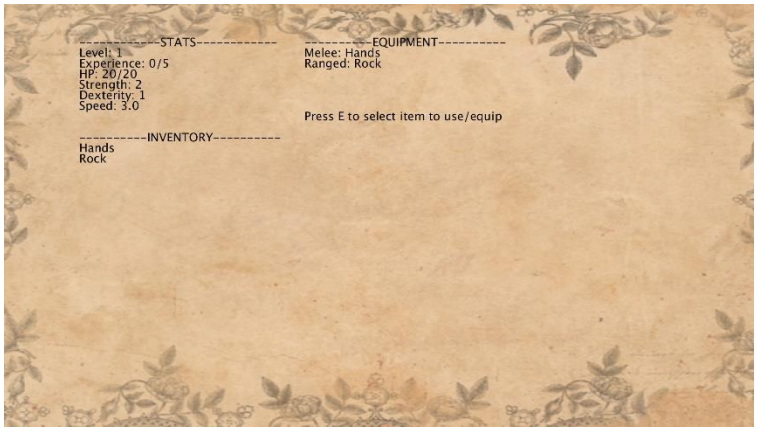
In level 3 and level 5, an additional weapon spawn can occur. In particular, in level 3 there is a 50% spawn chance of either a ‘Steel Sword’ or a ‘Bow’ to be placed in the level while there is a 25% chance for the ‘Zweihander’ and an additional 25% chance for the ‘Gun’ to be placed in level 5. The player might consider fully clearing the final level before escaping if they are lucky enough to get these drops.

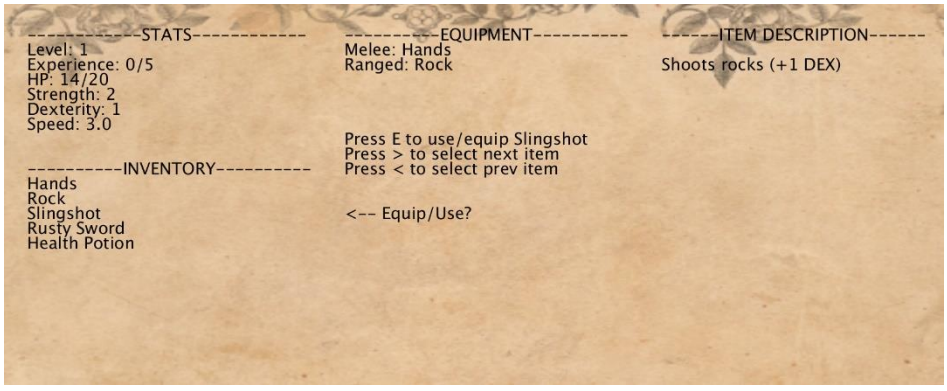
Screenshots of the game

• Game start

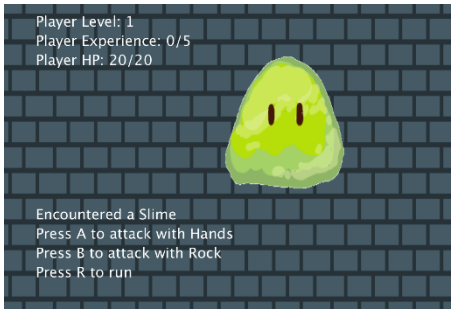


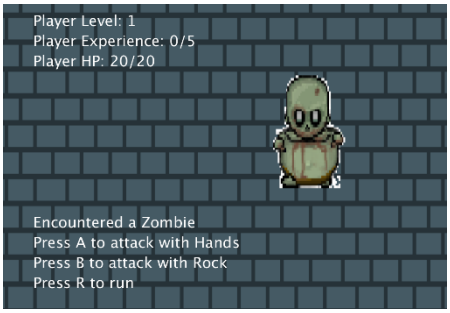
• Press S to bring up status page

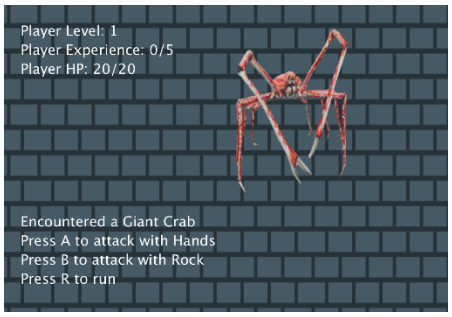


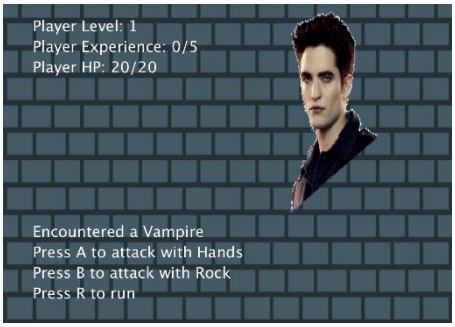


• Mob encounters



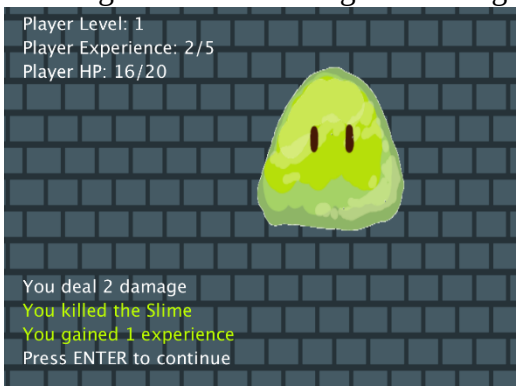


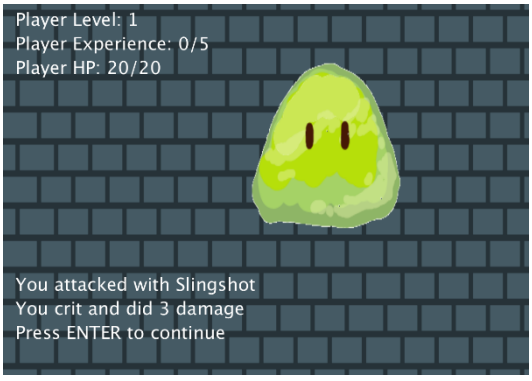




• Dying

• Killing a mob and critting with a ranged weapon





• Victory screen

• Demo: https://www.youtube.com/watch?v=N0z3ev5upRE

• Evaluation:
In this practical, the Roguelike was successfully implemented using procedural generation and AI concepts. The dungeon layout was procedurally generated and some mobs had decision making (e.g. Vampire). Mobs were placed in increasing difficulty to provide a sense of progression. An inventory system and items have been implemented and a treasure item to collect to increase the final score of the player. Players and mobs use kinematic algorithms to move and mobs have an additional steering algorithm when close to the player. Collision detection was implemented for all items, exits, mobs and the player. A battle system was implemented which paused the game during the sequence. Player stats were created and a status and inventory screen was implemented which also pauses the main game. A restart function is implemented when a player dies or when they escape. A running function is also implemented with a timer to ensure players don't immediately get into battle again. Mob attacks are based off dice rolls. There are prompts to guide players. The implementation was sufficient for a baseline roguelike.

• Conclusion:
In conclusion, the program was able to play a sufficient baseline roguelike. Additional features that could be implemented are a proper intro screen, more advanced damage calculator, a better attribute system and better mob movement.