Candidate no: 180000870

Introduction

The implemented roguelike game describes a wizard's journey through a procedurally generated dungeon as they grow in power, and face increasingly powerful and numerous enemies. The dungeon is a tower the wizard is climbing as they test themselves.

Design and Implementation

Designing the Roguelike

Game Balance design decisions
Game balance is essential to any game, and is primarily measured for this roguelike in how combat plays out, and the number of health potions the player has access to. In combat the player should defeat an enemy of the same level in approximately 3 turns, while the enemy should not defeat the player before then, and potions should be available to reverse most of the damage done.

Character Death
For the purpose of creating a true roguelike death is permanent, and nothing is carried over between characters. Allowing progression between deaths would be an interesting extra feature, but would also make game balance more difficult.

Scoring
The game over screen gives the character's level as the score. This is because there are finite enemies per level, and so a player must both defeat enemies and travel deeper into the dungeon to increase their level. Therefore their level is proportional to their performance.

Dungeon Level Design

I use the space partitioning method for creating my dungeon levels. The space partitioning method was chosen because it gives control over room size, while also accounting for walls around the room, the rooms cannot overlap and appear orderly, and the partitioning tree can be reused to group ‘territories’ for monster spawning and patrolling. The ordered appearance of the rooms is good because it better suits a tower climb aesthetic.

Space Partitioning, Agent based, Cellular Automata and Grammar based methods were all considered for the game. In order to produce a well ordered and logical layout (with efficient use of space) I chose not to use Cellular Automata or Agent based methods. The Grammar is interesting because it allows significantly more control over level layout, but requires too much effort given the time constraints, so I chose to use Space Partitioning. See the Appendix for the full notes.

Corridor Generation
Corridors are generated using the space partitioning tree, which is very useful because it guarantees that rooms in the same territory are directly connected. Corridors are drawn in straight lines because this suits the orderly aesthetic. By connecting each pair of children in the partitioning tree I guarantee that every room is reachable from every other room.

Dungeon Level Difficulty
Dungeon levels are the primary way I manage difficulty, by increasing the size of the dungeon and therefore the number of enemies. I chose this method because the larger dungeon increases the amount of loot for the player, so the increased effort is proportionately rewarded.

Movement
Both character and monster movement uses a kinematic algorithm. This is because the kinematic algorithm is more suitable for a game with tight corridors and small rooms, allowing the player and monsters to be controlled intuitively and moved with precision.

User Controls
The user has the standard ‘wasd’ controls common to PC games, with ‘e’ and ‘q’ allowing them to rotate, and ‘f’ used to interact with the environment. This setup allows the game to be played with one hand resting naturally.

Collision Detection
There are two components for collision detection. The first is preventing the player and monsters from passing through walls, and the second is detecting whether the player is close enough to a monster or item to interact. Whenever a wall collision occurs the player or monster is moved back into room space which appears as if the character is unable to pass a solid object. The collision between players and monsters checks the distance between the characters and triggers when a collision is detected, which causes a combat encounter.

Attributes
The design for the character attributes are:
Stats:

- Constitution
    - Increase max health

- Dexterity
    - Dodge chance
    - Accuracy
- Strength
    - **Monster-Only**
    - Damage
- Intelligence
    - **Hero-Only**
    - Player Spell damage

These stats were also picked because they could be used to resist status effect spells cast by the player, such as ‘blind’ or ‘weaken’. However, due to time constraints these spells were not implemented.

The player and monster used different damage stats for thematic reasons, but as a potential expansion of the game the player would have a strength stat for melee attacks, and some monsters would be able to cast spells and have an int stat.

I considered allowing dexterity to modify the movement speed of characters, but this would add an extra element to the game balancing because enemies may become too difficult to escape, or too easy in the dungeon level.

Item and Spell ranks
Each item and spell has a rank from 1 to 4. This is used when determining the strength of the item’s effect, e.g. a rank 4 healing potion fully heals the player while a rank 1 potion only heals 25%. Similarly higher ranked equipment grants larger attribute bonuses, while higher ranked spells deal more damage. However, higher ranked items are significantly less likely to spawn, and spells have an increasing cooldown equal to their rank.

It is future work to also tie Item and Spell ranks to the dungeon level, such that items at lower floors are better. This would reward the player for exploring deeper areas of the dungeon.

Equipment
Equipment is only implemented for the player. The player can wear a hat, a robe and a staff.
Equipment:
- Hat
    - Increase dexterity
    - Small increase to intelligence
- Robe
    - Increase Constitution
    - Small increase to intelligence
- Staff
    - Increase base damage
    - Small increase to intelligence

The player’s equipment and the different ranks they will encounter grants another way for the player to grow in power and progress. It is future work to implement additional

randomised bonuses to high ranked equipment to incentivise looting after the player has a rank 4 item.

Treasure
It was decided not to include treasure as an item because it has no mechanical value and I wanted the game to be about the growth of the character, not their items. However, if a system is implemented for carrying over progress between characters then treasure would be a useful way to acquire currency that may be spent on upgrades before the game begins. This would also be an interesting challenge because treasure takes up slots that could be used for potions or equipment.

Potions
Potions are the only way for the player to regain health in the dungeon, so their frequency must be balanced relative to the health the player is expected to lose per fight.

Inventory
The inventory is used to store potions and equipment for the player. It has a generous capacity of 10 because the higher frequency of low Rank potions means I expect the player to need several, in addition to their 3 items of equipment. It can be accessed, and the items interacted with, at any point in the dungeon including in combat.

Spells
Spells are the player's method for dealing damage in combat. While higher ranked spells deal more damage, they also have much longer cooldowns which incentivises keeping low ranked spells to use during the cooldown period. Three types of spell are implemented, but all are functionally identical. It is future work to implement elemental resistances to incentivising keeping a variety of elements, and equipment based bonuses to a type of elemental damage. Further it is also future work to implement different spell types, such as spells which disable and debuff the enemy, and spells that buff the player.

Spell cooldowns
Spell cooldowns are important to game balance because I wanted the ranks 2 to 4 to all be viable at the later levels. To accomplish this only rank 1 and 2 spells can be immediately cast at the start of combat, while rank 3 has a 1 turn cooldown, and 4 has a 2 turn cooldown.

Spell acquisition
The player is limited to 4 spells known, with the oldest spell being replaced when a new spell is picked up if no free slot is available. In order to better inform the player of what tier they are collecting (necessary, because they do not have total control over which spell is replaced) I use a different image for rank 1 and rank 2, while rank 3 and 4 share an image. This is because the lack of total player control introduces an element of randomness and challenge to the game.

Monsters
The dungeon is home to two types of monsters, Kobolds and Goblins. Both creatures use an AI based on pack behaviour where they will try to summon reinforcements before attacking. The Kobolds are stronger, but the Goblins are more numerous. The territories in the partition tree are used to group nearby rooms to form uninterrupted territories for either the goblins or

kobolds. While all monsters will pursue the player outside of their territory, they will only roam inside it, and will return if the player escapes.

It is future work to implement interaction between different types of monsters, and to add more monster types.

Pack Monsters
Both the Goblins and Kobolds use the same ‘pack AI’. This can cause the player to be overwhelmed by consecutive fights without the chance to acquire more potions, but can also be exploited by escaping before reinforcements arrive. These dynamics introduce more complexity to the game, requiring the player to pick their fights more carefully.

It is future work to reduce the player’s view to their close surroundings, so when they encounter a monster they don’t know if or when reinforcements will arrive, and must guess based on the nearby monster’s behaviour.

Monster Level
Monster levels are a secondary way I manage difficulty, by increasing the average level according to the dungeon depth. Variety is added to the strength of monster’s in a floor by randomising their levels slightly.

Combat
In combat the monster will either dodge or attack. The player will attack, dodge or attempt to flee.

If the player chooses to attack then they must choose a spell which is ready. Damage is then calculated depending on the rank of the spell, the player’s intelligence and the player’s weapon modifier.

If the monster chooses to attack, damage is calculated similarly, but without the spell modifier and using its strength instead of intelligence.

When attacking the enemy has a chance to dodge based on their level and dexterity versus the attacker’s level and dexterity. If the enemy chooses to spend their turn dodging then the dodge chance is calculated without considering the attacker’s level and dexterity.

Dodging is an interesting mechanic for the monsters because they may avoid a spell with a long cooldown. It is future work to implement heavy attacks with cooldowns for the monsters to incentivise player dodging.

The player may choose to dodge if the spell they wish to cast is on cooldown, and they want to minimise damage until it is off cooldown.

If the player chooses to flee they have a 50% chance to succeed. This value is not based on stats to avoid situations where the player wants to flee a stronger enemy, but can’t due to stat differences.

During combat the player may freely interact with their inventory without using a turn. This allows them to use potions, but the scarcity of potions relative to the number of enemies means the choice of when to use potions is still a challenge.

Implementation Details

Implementing game balance
Game balance was implemented by managing several factors of the game primarily through character stats, and the spawn frequency of useful items and monsters in a room.

Implementing Procedural Generation and Territories
The maps were procedurally generated using the space partitioning algorithm described in lectures. This is implemented by building a tree of nodes, where each pair of children further subdivide the area of the parent. Divisions are alternatingly by width and length to improve the chance of rooms appearing more square. Further each split has the chance of the children becoming ‘territory root nodes’ which means all of their children (which do not themselves become territory root nodes) are considered as part of the same territory for monster spawning and patrolling).

It is future work to implement more types of territories than ‘goblin territory’ and ‘kobold territory’, which will influence the number and types of items spawned in them, such as ‘treasure room’ territories. Further the type of territory can be dependent on the parent territory, so ‘treasure room’ territories may only spawn in ‘kobold territory’ areas.

Implementing Pack AI
The pack AI triggers on encountering a player. First it looks for other monsters of the same type in the area, then it tries to call them to attack the player, then it moves towards the reinforcements until it has a nearby ally, then the group attacks the player. Although relatively simple, this procedure gives the appearance of pack mentality to the Kobold and Goblin creatures.

I considered having the pack monsters flee when isolated, but this could be annoying for the player because killing monsters for experience is a core part of the game.

AI pathing
AI pathing is implemented using the A* algorithm as described in lectures. A search limit was imposed to avoid a single monster significantly impacting game performance. This limit is important because monsters can detect the player, and be alerted to their position through walls, so the actual path to the player could be prohibitively far.

It is future work to implement a more realistic method for player detection. The intended method is checking euclidean distance, then if the player is within a limit trying to path to the player using A* and a low search limit. If a path to the player is found then the player is nearby and it is reasonable to imagine they were heard or seen because the path is not completely blocked.

Improving Game Performance

As the player descends there will be more monsters on the map. The decision making for all of these monsters can be expensive, so I added several cooldown timers which force the monsters to wait before performing more expensive procedures such as pathing with the A* algorithm. Further 64x64 textures for floor tiles were added, but using them on the smallest levels still produced noticeable slow down, so they were removed.

Conclusion

The game meets the requirements of the specification, and is playable. However, due to time constraints several features, such as different spell types and a larger variety of enemies, were left out. As a result the game lacks the complex systems which encourage players to continue replaying roguelikes.

Appendix

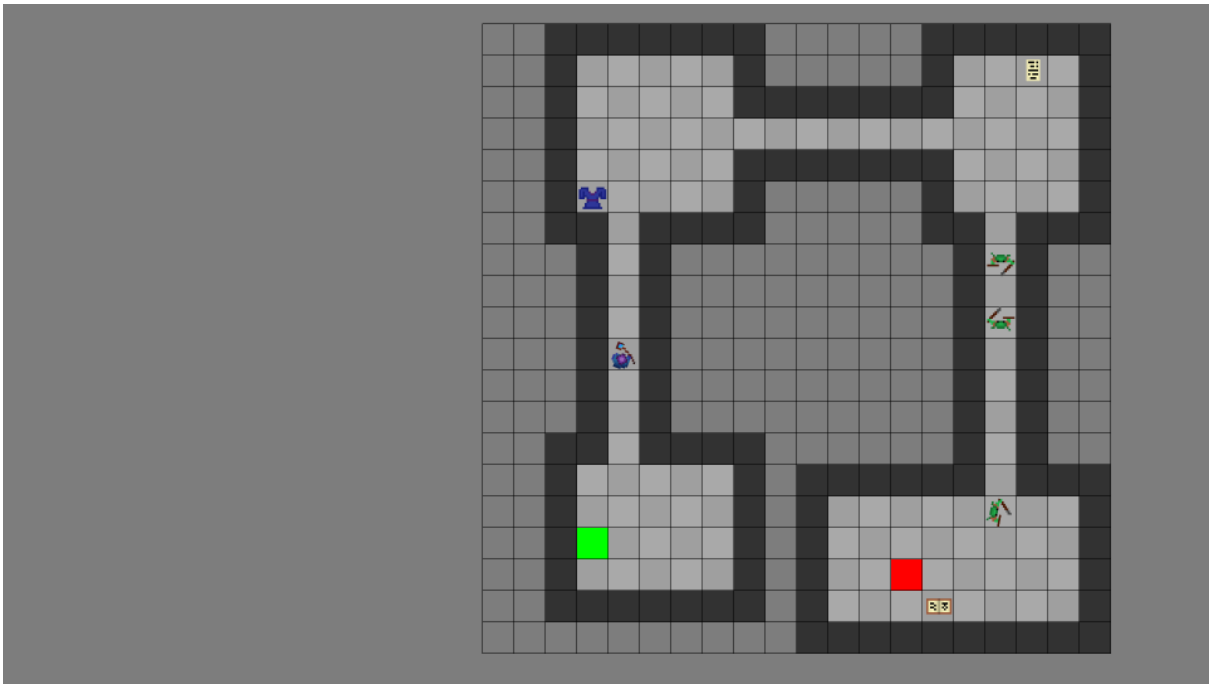
Full Content generation evaluation

Procedural content generation:

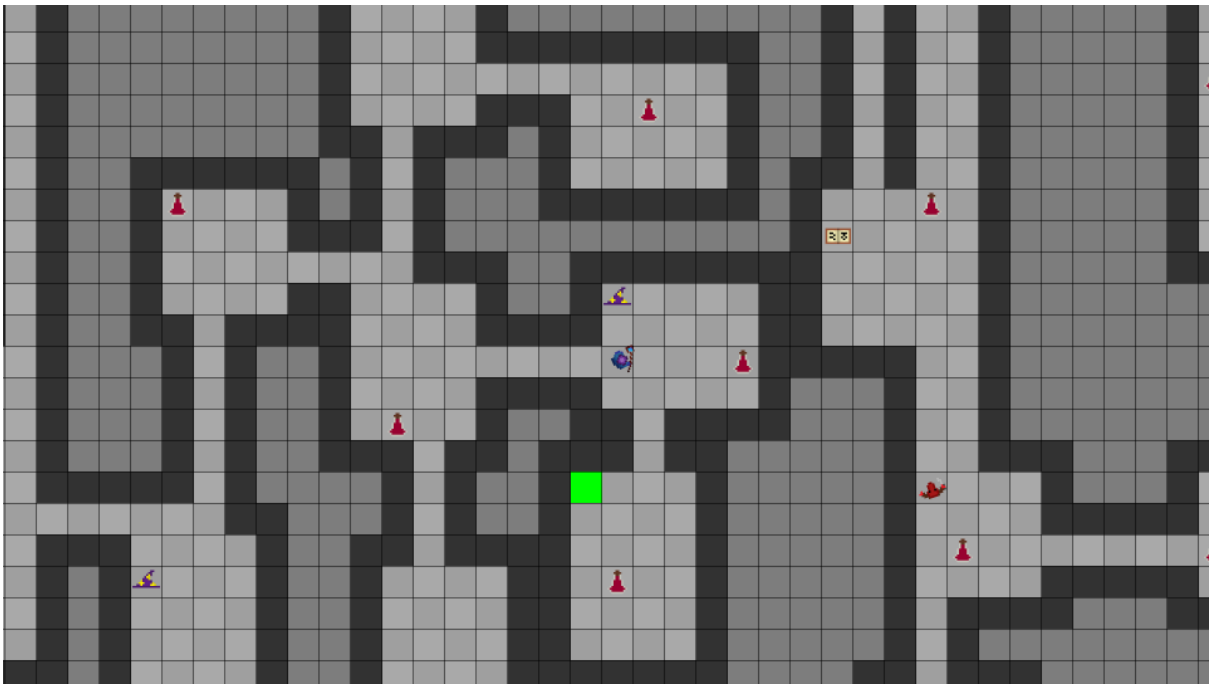
- Four techniques considered. Space partitioning, agent-based methods, cellular automata, and generative grammars.
- Space Partitioning:
    - An approach which avoids overlapping rooms
    - Well suited to regularly sized rectangular rooms
    - Creates neatly ordered rooms
    - Hierachy allows groups of proximal rooms to be grouped with a theme, e.g. overgrown, or flooded
        - And to group enemy types
- Agent based:
    - Micro approach to dungeon building without considering whole space.
    - Leads to a organic and chaotic dungeon
    - Can lead to overlapping rooms, and corridors if not using an informed look ahead digger
- Cellular-automata
    - Useful for producing organic, natural seeming environments.
    - Not possible to create a map with specific requirements (Togelius et al. 44)
- Grammar based dungeon
    - Used for generating a high-level topological representation of a level
        - Can use parameters such as global size
    - This approach is designed to improve control of gameplay for procedurally generated content
- **More notes:**
    - The PCG used is generic, not adaptive. This is because this approach is simpler, and supports the intended idea that the dungeon is inanimate, and exists independently of the player

    - Constructive PCG is used because it is more suitable for roguelike games (Togelius et al. page 9). Generate-and-test has a non finite upper bound of iterations needed, and is avoided because of this.
    - Considering using an agent-based method, or generative grammars. Agent-based method is generally simpler and produces more natural cave-like dungeons. The Generative grammar approach is more complicated, but permits
- Use **space partitioning** bc Grammar based is too complex, and agent based and cellular automata default to more natural room types, and space-partitioning allows creating multi-room areas.

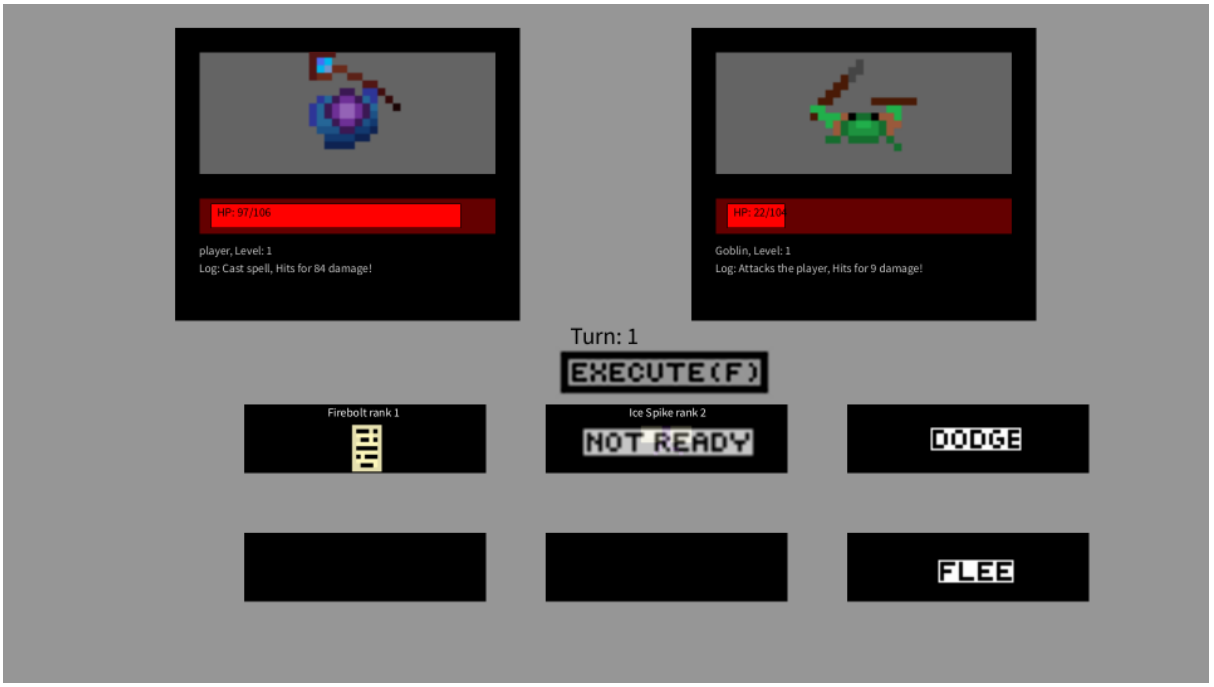
A dungeon first floor



A dungeon tenth floor



A combat screen



The inventory screen

Future work Notes

Combat:

- Wizard theme
- Cast spells, use items, flee
- Flee: if successful enemy cannot move for x seconds, player can attempt to escape
- Cast spells
    - Window of 4 slots
    - Each can be filled by a spell
    - Spells can be swapped out
    - Each can only be used in combat
    - Some are passive effects, some are active
    - Damage spells have different types:
        - Melee/Ranged
        - Ice/Fire/Storm
        - Instant/Damage Over time (uses 2x turn scaling, see control spells)
    - Active spells have cooldowns
    - Control spells have different effects:
        - Paralyse for 2x turns (scaling with Rank)
            - Targets Constitution
        - Sleep (until attacked) for 3x turns (scaling with Rank)
            - Targets Constitution
        - Blind (reduce accuracy by 50%) for 2x turns (scaling with Rank)
            - Targets Dexterity
        - Enervate (set Strength to 50%) for 2x
            - Targets Strength

    - Control spells have x% chance to be resisted based on user int, and enemy stats. If resisted the duration is halved.
    - Passive spells (Always on, but takes up a spell slot)
        - Grace +Dex
        - Fortify +Con
        - Insight +Int
    - Utility Spells:
        - Heal (restores (Rank%)/2 Health)
            - Can't miss
    - Spells have ranks:
        - Each rank is decreasingly likely to occur

Rank	Effect Scaling	Cooldown
Apprentice	25%	1 Turn (can be cast every turn)
Adept	50%	2 Turns
Expert	100%	4 Turns
Master	200%	8 Turns

        - Damage spells scale in damage
        - Control spells scale in:
            - num turns (4 * rank %, Apprentice x=1)
            - chance to overcome resistance (TODO)
- All attacks and spells have a % chance to miss based on (Strength/Int) compared to target dexterity
    - Control spells have 2 chances to fail because they are intended to be very powerful

References

Togelius, Julian, et al. *Procedural Content Generation in Games*. Springer International Publishing, 2016. Accessed 7 March 2023.