CS4303 P2: Roguelike

Hantao Sun
University of St. Andrews
St. Andrews, the United Kingdom
hs99@st-andrews.ac.uk

ABSTRACT

UPDATED—25 October 2018. This report describes a game including various elements of Roguelike games, which is implemented and designed basing on Processing.

Key word: Roguelike, Processing.

INTRODUCTION

After giving a name, players would be enabled to move in a dungeon. There are some options to get scores: they could choose to find treasure, collect the small pieces of chips, fight with monsters guarding treasure or step into higher level of mazes. There is not a standard to "win" the game, the target of players is supposed to be to gain more scores.

Players has 7 basic attributes: health point (HP), magic point (MP), attack force (ATK or Attack), defence force (DEF or Defence), critical chance (CRI), hit point (HIT, relating to the rate of hitting the target), dodge point (DOP, relating to the rate of being hit). These attributes would be influenced permanently by the level of players and some items or equipment, or briefly by some buffs or debuffs in battles.

There is also spell systems, which would detailly described in later parts. Besides, this report will also indicate the implementation of maze generation, interface, movement of players, AI of monsters and battle system, menu system and show more details about design of the game.

DUNGEON

The dungeon is composed of mazes of different levels

Generate

Implementing the Prim algorithm, all mazes of the game are generated randomly. In detail, the first step is to create a n * n matrix, assuming that they are n * n rooms. Then create another (2n – 1) * (2n – 1) matrix, representing the maze containing rooms (set as 1) and an obstructed road (set as 0) between every two rooms.

At first, all rooms are unvisited. From the entrance, trying to find an unvisited room next to the current room. If successfully find it, set this new room as current room and dredge the road between two rooms (change the road to 1). If not, randomly find a room from visited ones as current room. Repeat this until all rooms in the first matrix have been visited. As the directions of dredging and the decisions made when it cannot find an adjacent unvisited room from current room are random, the maze generated with this method is also randomized.

The complexity of the maze is decided by the level of the maze.

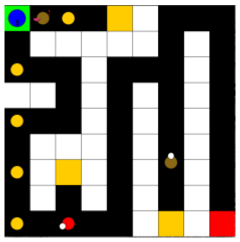


Figure 1. The level 1 maze

To simplify the design, the entrance (the green square in Figure 1) and the exit (the red square in Figure 1) are always at the top left corner and the bottom right corner of the maze respectively to avoid they are too close. Players could press ENTER at the entrance or the exit to move downstairs or upstairs. The mazes, which are created as objects, are saved in a list, every time players reach a new maze, it would be added to the list and these mazes would be read from this list when the players want to return to a visited maze. This could avoid regenerate and make it possible for players to collect left treasures or beat residual monsters.

Reaching a new maze, players could gain 5 scores.

Treasures

Treasure is a vital element for a maze game. In this game, treasures are represented by some yellow squares in maze. The number of treasures is positively related to the level of maze. Every treasure is guarded by a monster and treasures are always located in impasses, those squares which could be accessed from only one direction and will never overlap entrance and exit. That means players have to kill the monster and get the treasure freely or find a great moment to steal the treasure before being found by the monster (this would be discussed more amply in monster part).

To get the treasure, players needs to touch the treasure (but not step onto it, which is judged by collision detection), and press ENTER, then the treasure will open, and players will be informed what he gets from it. Certainly, a treasure could not be opened twice or more.

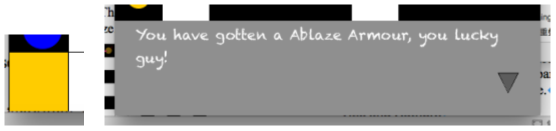

You have gotten a Ablaze Armour, you lucky guy!

Figure 2. The treasure waiting for opening(left) and the hint box shown after opening it(right)

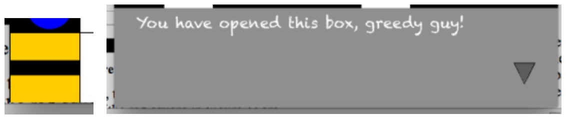

You have opened this box, greedy guy!

Figure 3. The treasure opened(left) and the hint box shown after trying to open it again(right)

The feedback of operations of players are mostly implemented by hint box and battle information (will be described in battle part). Players could not do anything when there is a hint box shown, until they press ENTER to remove it.

Opening a treasure would make players get 3 scores.

Chips

As it is a possible option for players that go to the exit directly from the entrance, without trying to beat monsters or finding treasures, some small chips are randomly set in each level of maze. Players could not reach next maze until they have collected all chips in this level. Impasses with no treasures would be preferentially chosen as the position of chips. Chips would not collide with the players, to make the movement of players smoother.

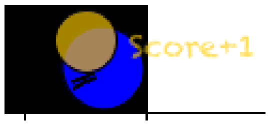


Figure 4. A player collecting a chip

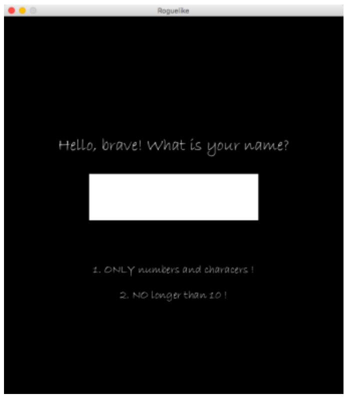
Players could earn one score from each chip, just like that shown in Figure 4.

PLAYER

In this game, players are represented by blue circles with the first letter of their name in it. The direction that the letter points to means the orientation of players.

Name

At the very first of the game, players will be asked to give a name to their character and cannot change it until they die and restart the game.


Roguelike

Hello, brave! What is your name?

1. ONLY numbers and characters !

2. NO longer than 10 !

Figure 4. Name interface

Movement

Movement is the most complex part of this section. Although movement itself is simple, the collision detection is difficult, because the size of the player is smaller than bricks in the maze. Detecting the collision between the player and all bricks in the maze in every frame is theoretically possible and easy to implement, but it is very time-consuming and will totally destroy the game experience. Hence it is necessary to generate simple detect strategies. There are two ways for players to move, with different collision detect strategies.

Move with direction

Using direction keys, players could move rapidly as they want. Processing could detect only one key pressed at the same time, so the player could only move horizontally or vertically and if collision happens, the position of the player would be fixed. As the result, the collision detect strategy is to detect the three locations near the location of the player, at the direction players are trying to move to. Just like those shown in Figure 6:

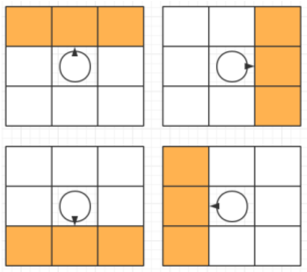


Figure 6. Collision detect strategy while moving with direction (orange locations are be detected)

Because the centre point of the player does not overlap the centre point of the square he locates at, it may performance poorly if only detect the brick next to the location of the player. If the three locations in this strategy have bricks or out of the maze, the player will be stopped.

Move with orientation

This mode of movement is implemented based on the kinaesthetic algorithm.

Just like that is mentioned before, all individuals, including players and monsters, have orientation attribute. Players could adjust their orientation by keeping pressing 'A' or 'D', and 'W' to move towards the direction of orientation.

Obviously, players could move in a velocity both horizontal and vertical, which means even though collision happens in a direction, players could also move in another direction as long as they have velocity in that direction. So, the program needs to detect those both in two directions, which is just like that shown in figure 7:

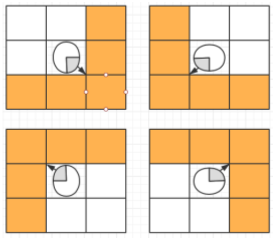


Figure 7. Collision detect strategy while moving with orientation (orange locations are be detected and grey range is the possible angle of orientation)

Specially, if the player only collides with the location that both in two sides, the change of the velocity should base on which side the player collides with this location.

Item & Equipment

There are 4 kinds of items: medicine, weapon, armour, shoes, while the last 3 kinds are also kinds of equipment. Each kind of items is divided into 3 levels. Item of higher level gives more help to the player. There are not many varieties and the functions of each item are fixed, so in design, getting and losing the item are implemented by changing the number of the item, which is not a very reasonable structure and could be refined in the future.

The parameters and attributes of item are stored and loaded through *Item.json* in */data* folder, making it easy to modify. While the player is fighting with a monster, players could only use medicines and cannot change equipment, which could only be done in the menu of item (menu system will be indicated later).

Players could obtain items by opening treasures and killing monsters, the level of gotten items are positively related to the level of the maze and the level of the monster. Specially, because there is temporarily no money and shop system in this game and the attributes of equipment are fixed, same equipment may be useless for players. Hence each equipment could be obtained once.

There are some details of functions of items.

Variety	**Attributes Increased**
Medicine	HP / MP
Weapon	ATK & CRI
Armour	DEF
Shoes	HIT & DOP

Table 1. Attributes increased by items

SPELL

There is a simple magic system in this game, including 6 elements, 3 levels, 4 types. In this system, there are six kinds of elements: Water, Fire, Wood, Earth, Holy and Evil. Each element has 2 spells in each level, and different feature.

- Water: Ability is average. Have all types of spells.
- Fire: Strengthening the ability of attack. All spells are to make more damage.
- Wood: Strengthening the healing ability. Almost all spells could offer healing power to the enchanter.
- Earth: Opposite to Fire, the defence capacity is improved. Almost all spells are to protect the enchanter.
- Holy: Having all kinds of buff spells. All spells are used to strengthen the attributes of enchanter.
- Evil: Having all kinds of debuff spells. All spells are used to weaken the enemy.

Each element has an independent level system for players. Every time players spout a spell of a certain element, they will get 1 point of this element. Getting enough points, players could unlock spells of higher level of this element. The number of points would also influence the real effect of spells, this value is calculated by:

Bonus value = Element Points * Bonus Rate (different for different spells)

According to the effect of spells, they are divided into 4 types: attack spells, healing spells, buff spells and debuff spells.

Attack spells

Attack spells could directly decrease the HP of the enemy. The real damage is:

Real Damage = Spell Damage + Bonus Value

Moreover, there are some restrict relations between elements, shown as Figure 7. The real damage is also influenced by the point of the element of the spell.

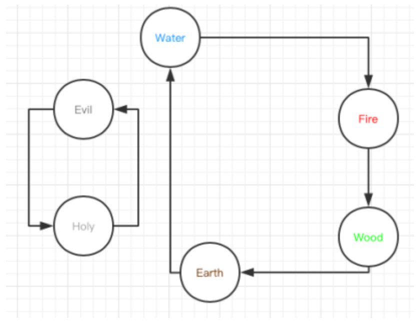


Figure 7. Restrict relations between elements

If the element of the spell restricts the element of enemy, the damage will become 4 / 3 times. The element of a certain kind of monsters is fixed, while it is decided the element with more points for players. For instance, if a monster spouts a Fire attack spell to attack the player and the player has more Wood element points than Fire, this situation would be regarded as restriction happens.

Healing spells

Healing spells could recover HP of the enchanter and only be spouted when the enchanter has taken some damage. The real healing:

Real Healing = Spell Healing + Bonus Value

HP would not bigger than max HP of the enchanter. Specially, healing spells could be spouted even though the player is not in battles.

These spells could make MP medicines much more precious than HP ones, so the regeneration of MP is limited to a lower value.

Buff spells

Buff spells could add a temporary buff to the enchanter, increasing a certain attribute or has special power. There are 7 kinds of buffs:

Variety	Attributes Increased / Effect
Attack	ATK
Defence	DEF
Hit Point	HIT
Dodge Point	DOP
Critical Chance	CRI
Invincible	Would not take damage
Growing	Recover HP roundly

Table 2. Buff effect

Debuff spells

Debuff spells could add a temporary debuff to the enemy, decreasing a certain attribute or restrict operations. There are 6 kinds of buffs:

Variety	Attributes Decreased / Effect
Attack	ATK
Defence	DEF
Hit Point	HIT
Dodge Point	DOP
Silence	Could not spout spells
Burning	Lose HP roundly

Table 3. Debuff effect

To initialize all spells, *Spell.json* is going to be loaded at the start of the game, just like data of items.

Finally, player or monsters could only spout spells when they have enough magic.

MENU

In the maze, players could press ‘I’ to call menus to check status, change equipment or recover HP by drinking medicines or spout healing spells. There are four kinds of menus showing status, equipment, item and spell respectively. Players could switch between different menus by pressing LEFT or RIGHT key. Players could press ‘M’ to return to maze.

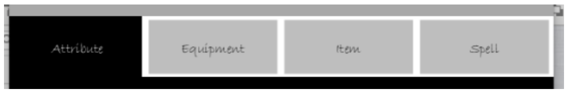


Figure 8. Menu switch panel (current menu is black)

Attribute menu

Attribute menu shows all kinds of information of the player, including name, HP, MP, level, experience and other attributes related to battles.

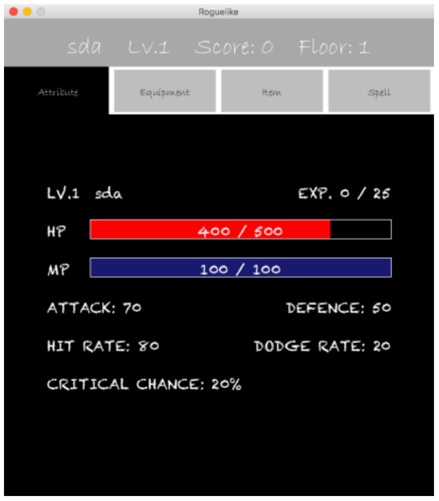
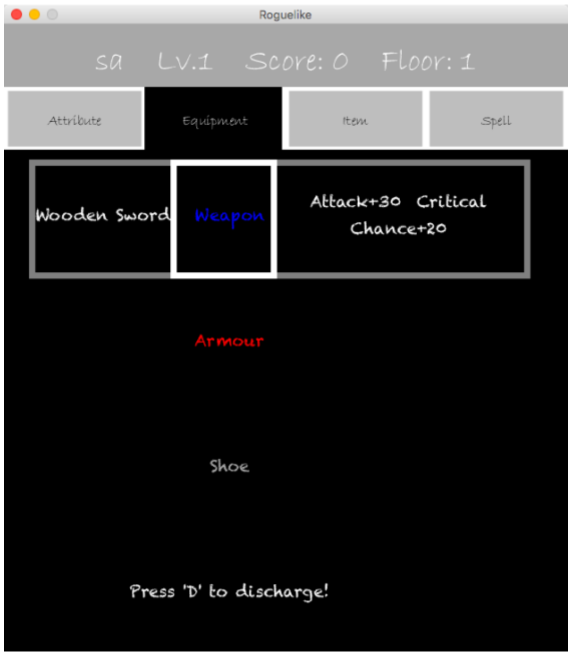


Figure 9. Attribute menu

Equipment menu

Players could check their current equipment and their effect.


Roguelike

sa Lv.1 Score: 0 Floor: 1

Attribute Equipment Item Spell

Wooden Sword | Weapon | Attack+30 Critical Chance+20

Armour

Shoe

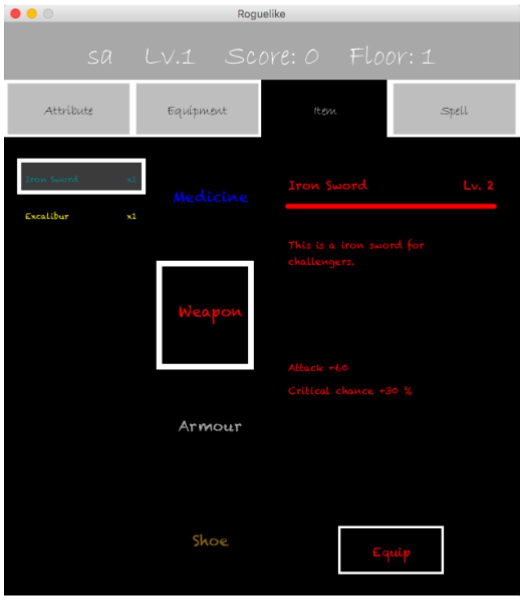
Press 'D' to discharge!

Figure 10. Equipment menu

Pressing UP or DOWN key, players could switch between different kinds of equipment or press 'D' to discharge the focused equipment. After being discharged, equipment would return to the item list.

Item menu

There are two levels of list in this menu. Player should firstly choose the variety, then choose the item to use or equip. All choosing operation is implemented by pressing ENTER, while players switch the focused option in all list by pressing UP or DOWN. Players could also press BACKSPACE to return to the last level of list (these operations are common in all later parts, so they will not be introduced again).


Roguelike

sa Lv.1 Score: 0 Floor: 1

Attribute Equipment Item Spell

Iron Sword x1

Excalibur x1

Medicine

Weapon

Armour

Shoe

Iron Sword Lv. 2

This is a iron sword for challengers.

Attack +60

Critical chance +30 %

Equip

Figure 11. Item menu

If players choose to drink a medicine when they do not need healing, there would show a hint box to tell players that operation is failed, and the number of medicines will not decrease.

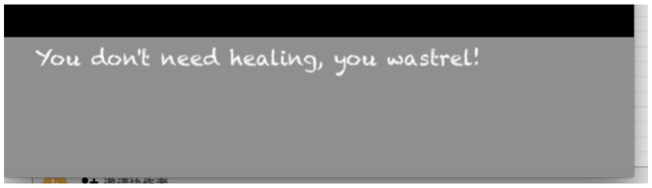
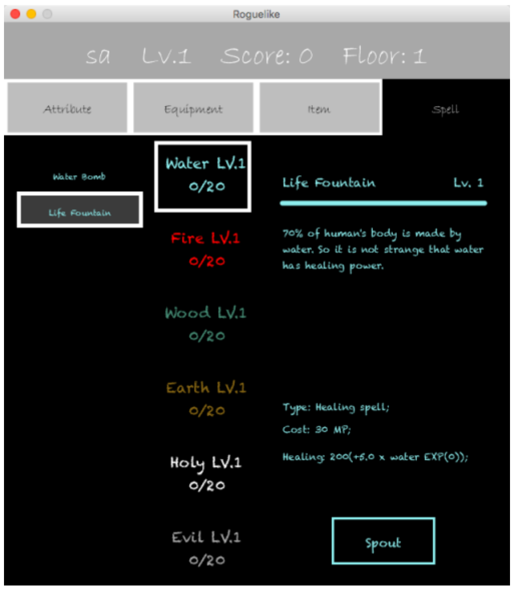

You don't need healing, you wastrel!

Figure 11. Hint of no need healing

If players try to equip something when they have already equipped this kind of equipment, the new one will replace the old one, logically equalling to discharge and then equip.

Spell menu

Technically, the spell menu is very similar to the item menu. Players need to firstly choose the element, then check the information of spells. Just like mentioned before, players could choose and spout healing spell in this menu, while other types of spells are unable to choose.


Roguelike

sa Lv.1 Score: 0 Floor: 1

Attribute Equipment Item Spell

Water Bomb

Life Fountain

Water LV.1 0/20

Fire LV.1 0/20

Wood LV.1 0/20

Earth LV.1 0/20

Holy LV.1 0/20

Evil LV.1 0/20

Life Fountain Lv. 1

70% of human's body is made by water. So it is not strange that water has healing power.

Type: Healing spell;

Cost: 30 MP;

Healing: 200(+5.0 x water EXP(0));

Spout

Figure 12. Spell menu

MONSTER

In this game, monsters are represented by coloured circles with circles or sectors marking the orientation of monsters. The colour of the "body" of the monster is decided by the element of the monster.

Monsters have some common behaviours with players, such as moving and fighting. That means they could extend a same super class and some method, like those controlling turning and moving, hence the movement of monster is also implemented with the kinematic algorithm.

Generation

Because of the limitation of time, there are only 2 races of monsters in this game: goblins and zombies, and the elements of them are Fire and Earth respectively. Each race has 3 levels of monsters: normal, leader and king. A monster has all spells of his element, whose levels are less than or equal to that of himself. For instance, the Lv.3 goblin king could spout Fire spells of Lv. 1~3. Other attributes such as ATK and DEF of monsters are written in *Monster.json.*

There is an interesting attribute of monsters: character. Even the same kind of monsters may have different characters. Character may influence the attributes and the battle AI of monsters. There are 3 characters in current design: rage, calm and timid. Each monster would be given a random character at the time of being created.

Table 4 indicates how the attributes of monsters are influenced by them. Details of battle AI will be shown in battle part.

Character	Attributes Influenced
Rage	ATK * 1.2, CRI * 1.2
Calm	DEF *1.2
Timid	HIT * 1.2, DOP * 1.2

Table 4. Characters and attributes influenced by them

Another two important attributes of monster are trophy, the item players get if they kill the monster, and experience value. The trophy is created together with monster, so possible trophies are only medicines, to avoid that players get duplicated equipment. Experience value is fixed.

Movement AI

As that mentioned in treasure part, every treasure is guarded by a monster. Hence the start points of monsters are always next to treasures, and the number of monsters of a maze always equals to that of treasures.

The behavior of monsters in the maze is based on a status machine like Figure 13.

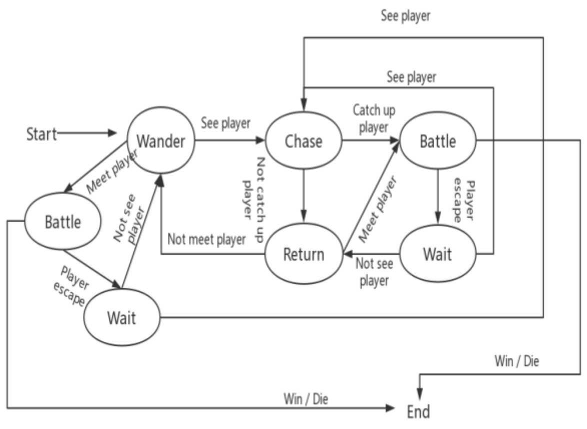


Figure 13. Monster behaviour in mazes

The status of the monster is marked by a String variable called *status*.

Wander

In normal status, the monster wander through a fixed path in the maze. The path is generated when the monster is created.

Because the maze is generated as a group of rooms or cells, the path consists of several rooms the monster will visit. The movement of wandering monster could be regarded as move from current room to next room. There are two status: “Come” and “Go”, used to mark if the monster is going to the end of the path from the start point or returning back.

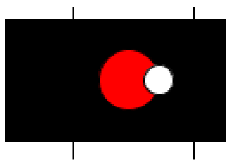


Figure 13. A wandering monster.

Detect and chase

If following three conditions are met, the monster will start to chase player:

- The distance between player and the monster less than 2 times of the size (width) of the cell of the maze.
- The monster orientates to the player.
- There are no bricks between the monster and the player. Because of the detect range mentioned in condition 1, there is up to 1 cell to be check.

As long as the monster start to chase the player, the current location and status would be saved. The chase path will be generated by recording the location of players. Since there will not be any loop route in mazes generated by PRIM algorithm, the process of chasing could be regarded as repeat the route of the player. There is another list of locations which will refresh in every frame to record the location of the player. The speed of the monster will keep increasing during chasing until it equals to or bigger than the max speed, making it possible to catch the player even though it needs to run further.

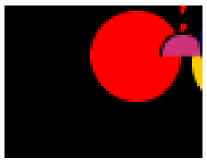


Figure 13. A chasing monster.

There is a problem: how to avoid the monster repeat the returning behaviour of the player? The solution in this game is that remove the last location saved in the list if the location of the player equals to the penultimate one. This algorithm is also suitable for mazes having loop routes. If at a moment the player is at the same cell as the monster, the battle will be triggered. It doesn’t utilize the collision detection because chasing is difficult to implement with the chasing algorithm used in this game.

Return

If the monster has not caught the player when it chases through 5 cells, the monster will start to return to its origin path.

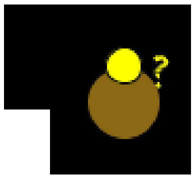


Figure 14. A returning monster.

This process could be considered as the inverse process of the chasing, and easy to implement, which could be done by tracing back the chasing path. The destination of the returning process is the start point of chasing. If the monster successfully returns back, the status will return to the status before chasing.

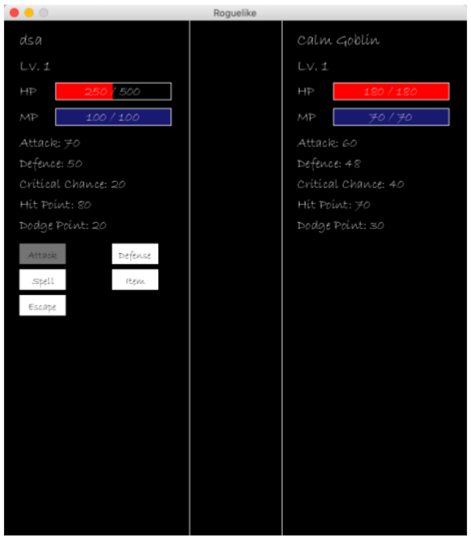
Wait

If the player escapes from the battle with a monster, the monster will wait in place for a short time to make player possible to run away. Otherwise, the battle will be triggered again. In fact, "Wait" is the only status in which monsters will never trigger battles with players.

The outlook of monsters waiting is as same as that of monsters returning.

BATTLE

Battle runs in an independent interface.



dsa
Lv. 1
HP 250 / 500
MP 100 / 100
Attack: 70
Defence: 50
Critical Chance: 20
Hit Point: 80
Dodge Point: 20

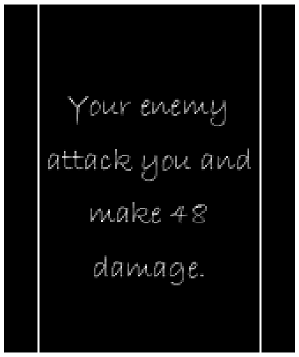
Calm Goblin
Lv. 1
HP 180 / 180
MP 70 / 70
Attack: 60
Defence: 48
Critical Chance: 40
Hit Point: 70
Dodge Point: 30

Figure 15. A chasing monster.

The battle is always started from the player 's turn and finished with death of one side. If the monster is killed, the player will get the trophy and some experience.

Battle information

The gap between two sides is used to show some instant information, including feedback for player ' s behavior and description to monster ' s behavior.



Your enemy attack you and make 48 damage.

Figure 16. Battle information

Behaviours

There are 5 choices for players in their turn, responding for the 5 buttons shown in Figure 15, and the same for monsters except using items.

Just like that mentioned before, all operations of users are completed by pressing direction keys to switch the choice, ENTER to choose or BACKSPACE to go back.

Buff & Debuff

Both the player and the monster have a list saving buff and debuff respectively. Created as objects, buffs and debuffs have 2 methods: *work ()* and *retire ().* The former method is to make the buff or debuff take effect, while the later one is to make it lose efficacy. At the start of every turn, the buffs and debuffs held by the owner of the turn decrease a round until they have no rounds and call *retire ()* method.

Most of buffs and debuffs take effect at the moment they are added, except "Growing" and "Burning", which are change the holder's HP roundly. So these two kinds of effect need to call *work ()* at the start of every turn of the holder before they lose efficacy.

If the battle finishes with death of monster or escaping of someone, all buffs and debuffs havn't losing efficacy will call *retire ()* method.

Buffs and debuffs held by player and the monster could be checked next to their level text.

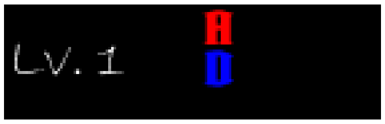


Figure 17. An Attack Buff and a Defence Debuff

Attack

The result of attacks is decided by ATK, HIT and CRI of the attacker, the buff list, DOP and DEF of the target.

In detail, attacks could be executed only if there is not a Invincible buff in the buff list of the target. After checking it, the game will calculate if the attack is missed. The hit rate is calculated by:

Hit Rate = HIT (Attacker) / (HIT (Attacker) + DOP (Target))

Then the damage of the attack will be calculated by:

Damage = HIT (Attacker) * 100 / (100 + DEF(Target))

Finally, if the attack is critical, which is decided by critical rate of the attacker, the damage will become 1.5 times, and if the target choose "Defence" at last turn, the damage will become 0.75 times.

Hit rate, critical rate will work by compare with a random number. If these number less than rates, they would be considered actually hit or critical attack. All uncertain results will be decided by this method.

Defence

Players could choose to defend themselves from next attack or attack spell pointed to them. The value of damage reduction is introduced in last part.

Spell

Players and monsters could spout spells to make damage, heal themselves, add buffs to themselves or debuffs to their enemy.

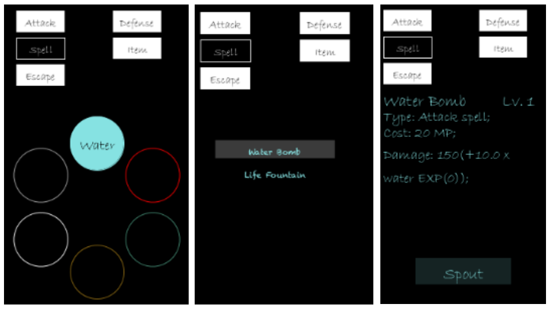


Figure 18. Spell interface (left: choose element, mid: choose spell, right: choose to spout or not)

Item

Only player could use items and the only type of items allowed to use here is medicine.

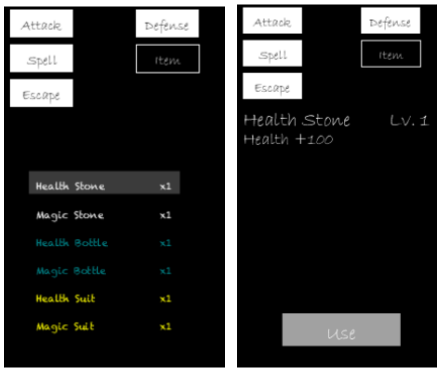


Figure 19. Item interface (left: choose medicine, right: choose to use or not)

Escape

Players and monsters could choose to escape from the battle. However, their escapes are not always successful. The escape rate is calculated by:

Escape Rate = (HIT (escaper) + DOP (escaper) / (HIT (escaper) + DOP (escaper) + HIT (enemy) + DOP (enemy))

If the escape fails, the battle will step into next turn.

Monster battle AI

The behaviour in battle of monsters is decided by a simple decision tree.

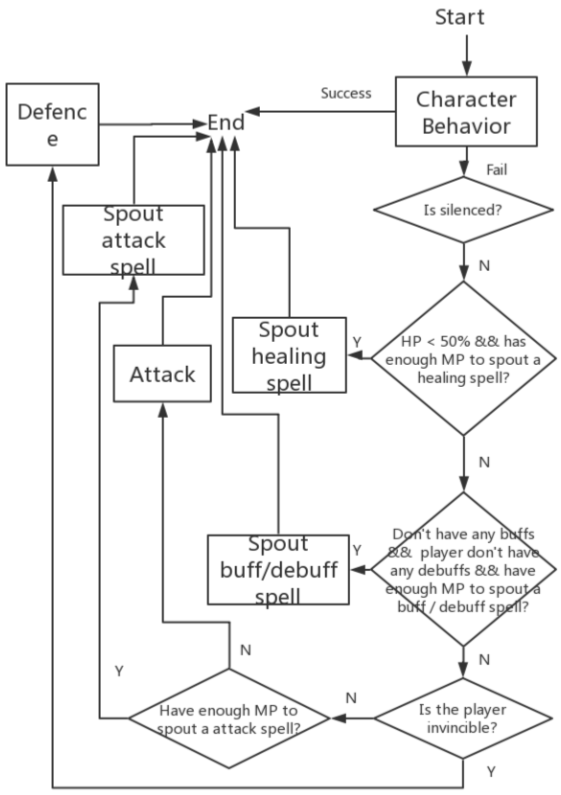

Start
Character Behavior
Success
End
Fail
Defence
Is silenced?
N
Spout attack spell
HP < 50% && has enough MP to spout a healing spell?
Y
Spout healing spell
Attack
N
Don't have any buffs && player don't have any debuffs && have enough MP to spout a buff / debuff spell?
Y
Spout buff/debuff spell
N
Is the player invincible?
Y
N
Have enough MP to spout a attack spell?
Y
N

Figure 20. Monster battle AI

There is a point need to be notice that it will firstly check if the monster is able to spout a certain kind of spells, then check if it has enough magic to spout a certain one among all choices.

Character behaviour

The character of the monster may influence its behaviour in the battle. More detailly, there is a small possibility (20%) to trigger the unique character behaviour at the turn of the monster. Characters and corresponding behaviours are listed in Table 5.

Character	Attributes Influenced
Rage	Attack
Calm	None
Timid	Escape

Table 5. Characters and corresponding behaviours

Especially, if the player is invincible and the character is "Rage", it will be considered that the character behaviour is not triggered. Moreover, if the monster successfully escape, player will not get experience and trophy from this battle and the monster will not appear in the maze again. So it is a good strategy to try to decrease the escape rate if the player meet a timid monster.

DEATH

If the player is killed by a monster, he could only restart another game and not reborn at a certain place. In other words, the death of this game is everlasting.

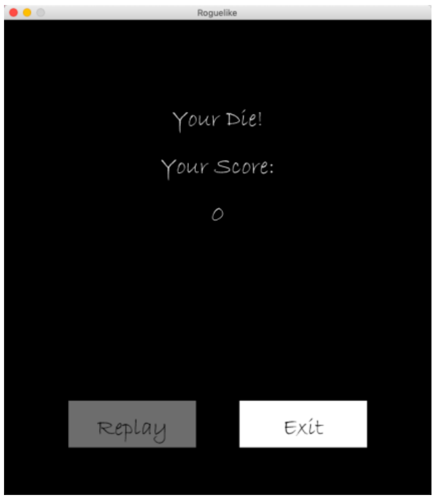


Table 5. Death interface

SOUND

Every time player switch options from a certain list or drinks a medicine, a spell is spouted, or someone attacks his enemy, there will be an audio played. Technically, this is implemented basing on the Minim library of Processing. By calling *rewind ()* and *play ()* methods of *AudioPlayer* objects, these sounds saved as mp3 files could be played conveniently and repeatedly.

SOURCES

All mp3 and font files are downloaded from the Internet.

CONCLUSION

Although this game has some essential elements of Rougelike, it is not a very good game. AI of monsters and UI design are still a little shabby, and there are supposed to be more systems to make the game more interesting.