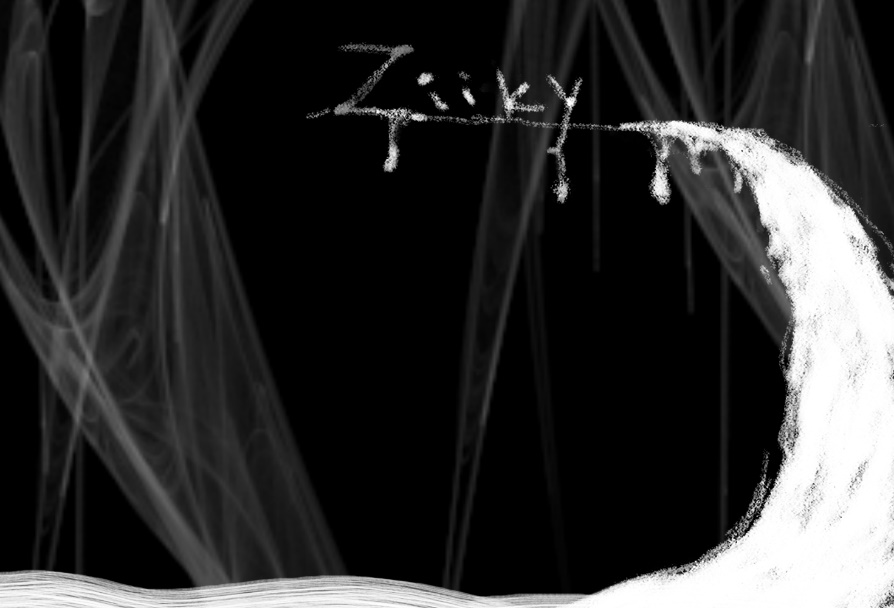
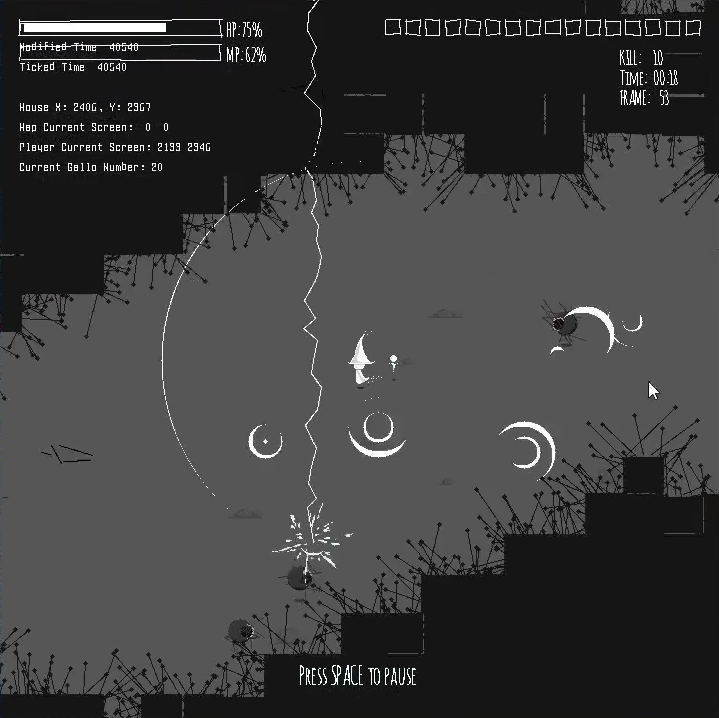
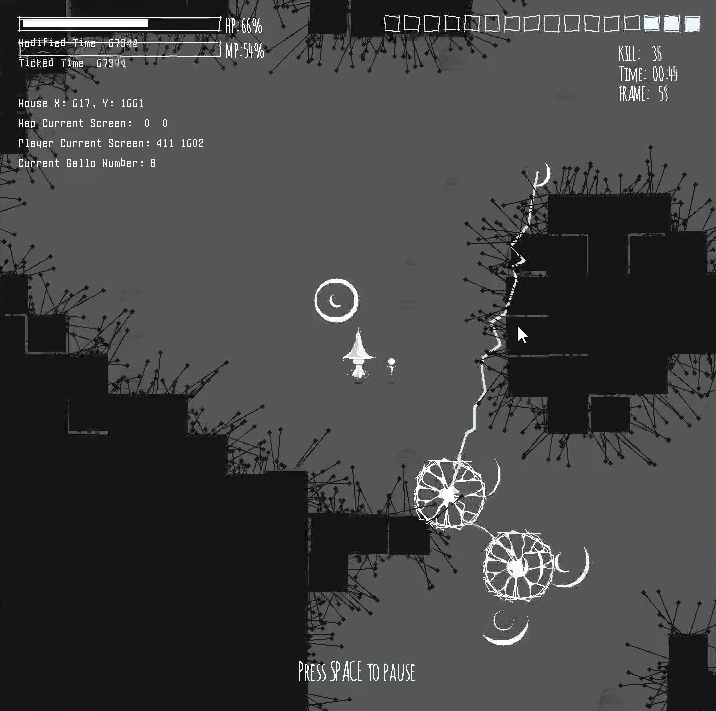
Name: \_\_\_\_Ziqi Yang\_\_\_\_\_\_. Student id: \_\_16521586\_\_

C++ Programming, Coursework Part 2, mark sheet

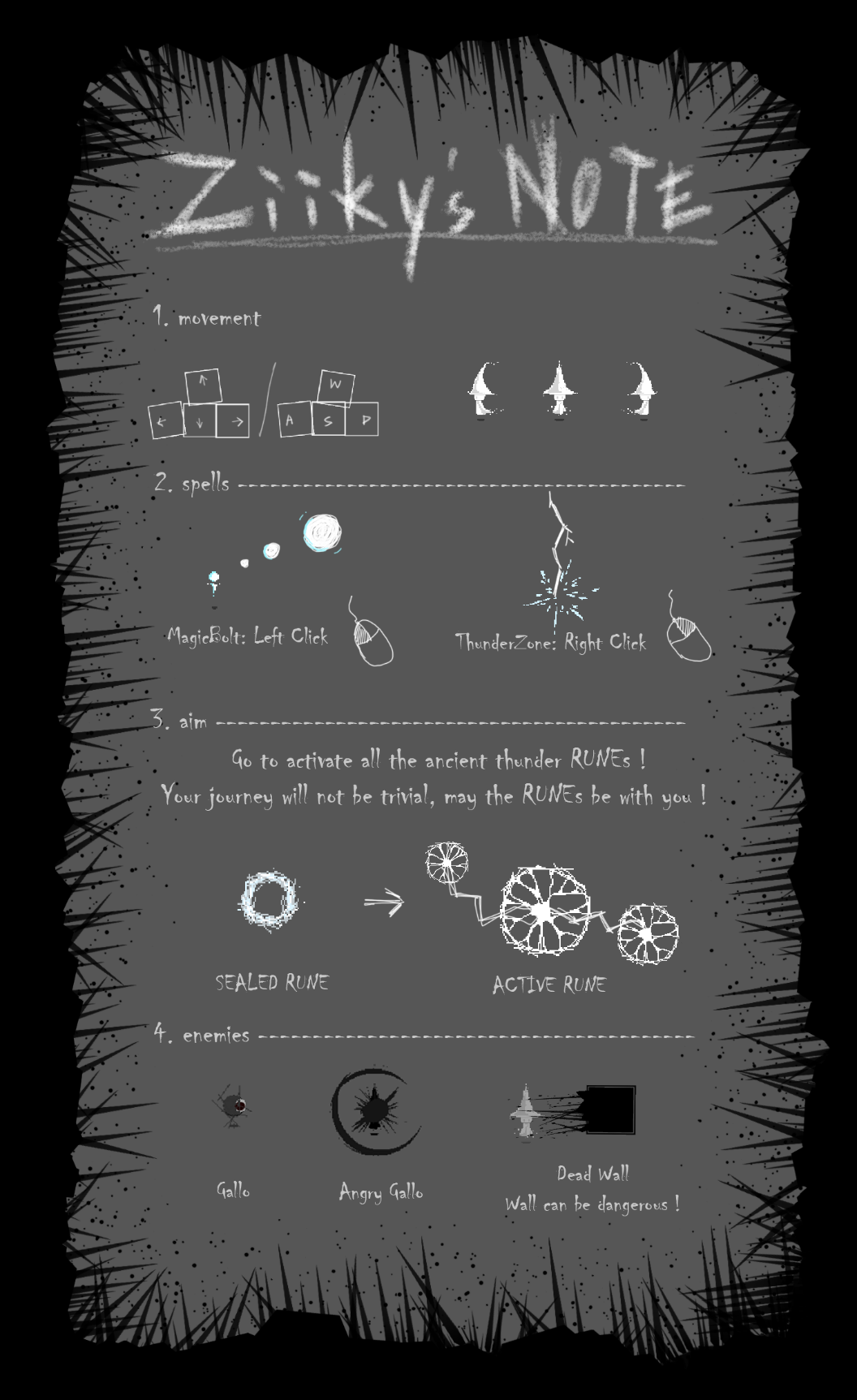
Screenshots

Start Menu & Concept Art

Loading Scene Game Over Scene

 Winning Scene Concept Pause Scene

**Game Demo Screenshots**



**Player Manual**

|  |  |  |
| --- | --- | --- |
| Mark | Requirement | **What you did and where it is in the code (file/lines)**  **You should fill in this boxes in this column** |
| 0-3 | **1. Add states to your program.**  Multiple states, different non-trivial backgrounds to demos  Advanced (3) = State Design Pattern | The **state machine**(Finite State Machine) receives jobs from engine and relays the job further to the current scenes (6 scenes: 1.MENU; 2.LOADING; 3.GAME(gaming, pause, win(inner scene)); 4.RESULT; 5. INSTRUCTION; 6. RECORD). It is state machine's job to determine and switch between scenes.  (see **SceneSystem** folder) (to check how BGs works, please see **MapSystem** folder for **GameScene** and **UI** folder for all scenes)  Classes:   1. **BaseScene**   The basic structure of each scene to handle different event such as onCreation, onDestroy, onActivate, onDeactivate…   1. **GameScene**   Where the player plays the game. Controls all the elements in the game: Map generator, enemy spawner, camera, environment render, UIs... It has three inner state as **Pause, Game over and Win, (the reason they are inner scene because they are designed as semi-transparent UI, where some interesting animations of the game are remained even in pause state (not all elements in the game have to be paused.)**   1. **LoadingScene**   A Fake Loading Scene because we are doing mono-threading, the loading cannot really load under the scene, this scene is only for hiding the loading time and giving players a professional feeling.   1. **MenuScene**   An animated Menu Scene with a sand simulation system interactable by mouse clicking (see below)   1. **IntroScene**   A zoomable and draggable scene containing user manual implemented by FilterPoint.   1. **ResultScene** & **RecordScene**   Two similar scene shows the player the record made and the best and handles record manipulation.  (Expects 3 marks) |
| 0-3 | **2. Save and load some non-trivial data**  1=one value save AND load  2=more complex data | In **result scene**(shown when player wins), a record board will be shown to the user having all the data he/she has in the current game (Runes Activated, HP remained, Surviving Time, Num of Kills) with comparing the best record ever read from **record.txt** file. Any record broken will be prompt and the user **can input his/her name and update the record file by all the records he/she broke, or the user can discard the record of the game this time.**  In the **record scene**(from Menu), the user can view all the best record(HP remained, Longest and shortest Surviving Time, Num of Kills) and **can reset the record file as default.**    What above is implemented by a **RecordManager** in the **base folder** under the scene. RecordManager is a **singleton** can be used by UI gameObjects to read/write from files.  (Expects 2 marks) |
| 0-1 | **3. Use appropriate sub-classing with automated objects**  Intermediate class with functionality.  At least three different subclasses. | There are too many end-class from gameobject in the project, what below are important intermedia sub-class or interface:   1. **GameObject**   All game objects inherit from **DisplayableObject** class and will be contained and handled in the **DisplayableObjectContainer**. Each game object is enhanced with a **sprite** and **ImageMappingGeneralist**(see below), which can display image with rotation, translation and colour Manipulation. Each GameObject uses **center as coordinate** and positions are handled as **Vec2**(see below) by default.   1. **Damageable (abstract)**   Specifying a game object that can be damaged by attacks/spells by **takeHit**(int damage).   1. **LivingEntity**   Subclass of Damageable and GameObject, specifying a game object that has a life and can be killed (such as enemies, player).   1. **Collidable** **(abstract)**   Used together with **CollisionSystem**, specifying a game object can collide actively to others.   1. **BaseButton** (UI)   Specifying a game object that is an **animated** (see Animator&Animation class) **button** which **can handle user mouse click/hover events.**  (please see all these classes in **Base/GameObject/Inheritable subfolder**)  (Expects 1 mark) |
| 0-1 | **4. Creating new displayable objects during operation**  Created or made visible | Approach Made Visible:   1. **Gallo** (the name of our enemy)   Inactive state: all dead/asleep gallos are inactive which are **invisible** and to be re-spawned by **EnemySpawne**r.  Sleep: when the gallo is far enough (1000 radius) from the player and think the player will not be approached in a while, it stops wandering and makes itself sleep and waits for the **EnemySpawner** to awake it at a new position and set visible again near the player.  (please see all these classes in EnemySystem folder)  (Expects 1 mark) |
| 0-1 | **5. Correctly destroy displayable objects during operation**  Destroyed, not just made invisible | **Rune**:  All the runes are initialized in inactive state by **RuneManager** during map generation phase and controlled during the game. When the player approaches a rune and activates it, the rune will go into activating state and release powerful spell to kill all enemies around(400 radius) and finally go into **active state delayed by 5 seconds**. After that this rune will no longer be needed and will be **removed** from the **DisplayableObjectContainer** and **deleted**.  (to check how the removing and deleting processes are implemented, please see **virtMainLoopDoBeforeUpdate**() function in the **GameScene**.**cpp** file from **SceneSyetem** folder)  (Expects 1 mark) |
| 0-4  \*  + | **6. Complex intelligence on an automated moving object**  1-2 marks then justify to the right ->  3 or 4 marks needs good documentation explanation later.  4 marks needs video! | (see section below)  (Expects 3 marks) |
| 0-2 | **7. Non-trivial pixel-perfect collision detection**  Improved=better than the class I gave  Complex irregular shapes is meant to be hard – to challenge the best students, e.g. check each pixel in shapes for intersection. | **For the 1st mark** :  I add an Enhanced **CollisionDetection** class with checkObstacleBySquare to check four corner of a center position instead of only check the radius. This can help a lot when the player are detecting the wall tiles and make it more precise and enhance performance.  **For the 2st mark** :  I designed **a particle system** as a **sand simulation** by **cellular automata algorithm** on the **Menu Scene** and **loading Scene(because I don’t need a complex perfect detection in my main game)**.   1. This is implemented by a tile manager contains the sand and empty tile types. Instead of loop from 0,0 to maxX, maxY, we need to loop backwards to simulated a falling physics. The basic **cellular automata mechanism** is to check if there is an empty slot immediately under the current particle, if yes, swap their tile type; if not, then check the tile at the left of the tile and then right.   **To simulate an advanced real world physics**, I add **an acceleration by gravity** and the particle with higher speed can then check and swap with the tile at lower position instead of the nearest one.   1. By clicking left mouse button, the user can generate more particles onto the scene; by clicking right mouse button, the user can erase all the particles around the mouse position. And all other particles will simulated the sand behaviour to fill the erased gap. 2. when the number of particles exceeds a specific number, the bottom level of particles will be eliminated to ensure there will not be too many of them! 3. Each particle unit is a struct contains a fixed colour to simulated a noise effect of sand instead of make them the same colour, which will not be obvious to show the effect.   What I did for 2nd mark was Inspired by a recent game published called Noita, the link is the talk about the hidden Tech of pixel particle system: <https://www.youtube.com/watch?v=prXuyMCgbTc>  (Expects 2 marks) |
| 0-2 | **8. Have advanced animation for background and moving objects**  Objects AND background  Proof of concept or smooth animation? | I have two types of animations:   1. Image-frame **animation** and **animator**   When a gameObject needs to display itself as aniamtion (with images), it is the **animator's** job to realize the current **state** of the game object, switch and display the animation accordingly. Each animator class is generated and initialized from template class code based on the state it will handle. E.g. player can have a player state with idle, running left, right and etc. Each animator has several animations according to different states.  Each animation has several images/sprites as a set of frames and controls the fps of the animation.  (to check this type of animation, please see **BG of MenuSecen, User manual of IntroScene, Smoking Effect in PauseUI, BG in WinUI, Player, Wand and all UI buttons**)   1. Procedural animation by code   In order to implement procedural animation and effect, I write some 2d & linear math function to use: **clamp**(with boundary), **random generator**(with boundary), **linear interpolation, ping-pong**, **smooth step** to play with the origin draw function provided in framework.  (to check this type of animation, please see **flying and explosion of Projectile, lightening strike effect of ThunderBolt, animation of Gallo, animation of DeadWall, camera shaking and fuzzy effect of GameCamera, and all UIs)**  (Expects 2 marks) |
| 0-2 | **9. Interesting and impressive tile manager usage**  **5+ tile types**  **Appropriate/different pictures**  **One or more images**  **2 marks: impressive** | 1. **CaveGenerator**   has tile type for generating the random map (see 18th req.):  cave\_tile\_empty = 0;  cave\_tile\_wall = 1;  cave\_tile\_edge = 2;  cave\_tile\_spawner = 3;  cave\_tile\_rune = 4;   1. **GroundRender**   This tile map as environment renderer will control the random environment features like fuzzy visual effect and different images of stones on the ground.   1. **Particle System** is also a tile manager 2. **DeadWallSpawner**   1. This type of enemies is treated as dangerous environment, which is undamageable and player needs to avoid  2. All dead walls are generated by **DeadWallSpawner** living on the edge of the cave.  3. When the player approaches, all walls around the player will be awake and start to attack the player at a short interval with lower damage (3).  (Expects 2 marks) |
| 0-1 | **10. Allow user to enter text which appears on the graphical display** | **ResultScene:**  Please see ResultScene shown when the player wins the game to handle keyboard input, delete and simple empty input error checking. To quick the win game, you can press “o” in game scene or hard-code to tell the FSM(statemachine) the first scene should be ResultScene in Zy21586Engine class.  (Expects 1 mark) |
| 0-1 | **11. Image rotation/manipulation using the ImagePixelMapping object**  Your own new subclass! | All game objects have a **ImageMappingGeneralist** object by default which can handle image translation, offseting, rotaion,hue, merging and brightness.  Please **see BG in PauseUI, Player sprite for transparency and brightness, Wand and Buttons of MenuScene** as examples.  (Expects 1 mark) |
| 0-1 | **12. Show your understanding of templates**  Either classes or functions.  Used in appropriate way**.** | 1. **Vec2** struct **from Utils.h** has template functions operator\* accepting arithmetic variable. 2. **Animator** is template **class** accepting enum. 3. **Some Utilis functions** **from Utils.h** are **templates functions.**   (Expects 1 mark) |
| 0-1 | **13. Show your understanding of operator overloading**  Not the = or == | 1. **Vec2** struct **from Utils.h** has three operator overloading \*/+/= to handle coordinate/2d math.   (Expects 1 mark) |
| 0-1 | **15. Use your own or standard smart pointers appropriately**  Evidence your understanding.  Used appropriately. | 1. **Animatior**   Because each **Animator** has a set of **animations** and controls which animation to play according to the current state of the owner game object. So I defined all animations of a animator as **shared**\_**ptr** so that I don’t have to manually delete all animations of the animator of the game object when deleting the game object and its animator.   1. **Projectile**   Projectile is not a game object but rather a **Collidable** class contains information of the projectile position, when to explore, if hit enemies or walls. It is the **Wand’s** job to render animation of each projectile based on their states. So I defined all projectile as **a vector of shared\_ptr** inside the Wand class. So the wand doesn’t need to delete them manually every time a projectile needs to be destroyed. They can be dynamically generated and destroyed easily. (simial mechanism in ThunderZone & ThunderBolt)   1. **Scene**   All scenes managed by FSM all saved as shared\_ptr as well.  (please see **Animatior** of **AnimationSystem** folder, **Wand** and **ThunderZone** of **SpellSystem** and FSM mentioned above in 1st Req.)  (Expects 1 mark) |
| 0-1 | **15. Correctly implement scrolling, allowing the user to scroll around using keys and/or mouse**  Do not need to use FilterPoints for this one – as long as it looks like scrolling using keys or mouse | 1. **GameCamera**   Camera class contains and handles FilterPoints - translation added on drawingSurface. GameCamera can be bind onto a specific game object (player in this game) and follow the game object while it moves. GameCamera will add a strict limit when player approach the boundary of the map, instead of place the player as center, it will be restricted by the map boundary. It can handle simple camera effect like shaking.  (Expects 1 mark) |
| 0-1 | **16. Correctly implement zooming, allowing the user to zoom in and out using keyboard and/or mouse**  You need to use FilterPoints class! (So need to understand that.) Need your own sub-class of filterpoints. | 1. **ViewCamera & Zy21586FilterPointsScale**   **ViewCamera** class contains and handles a **Zy21586FilterPointsScale and a FilterPointsTranslation.** ViewCamera is used in **introScene** to zoom and shrink the user manual(Dragable gameobject) by keyboard (w to zoom, s to shrink and space to reset) and mouse (click to drag).  (Expects 1 mark) |
| 0-3 | **17. Impact/impression/WOW factor!**  2 or 3 marks needs explanation in document.  3 marks needs a short video too. | I think my game is, though not an outstanding nor with a fascinating mechanism, playable enough and I can proudly say that I definitely put significant effort on this work, both on coding (nearly 20k lines of code) and art concept. Hopefully it would be impressive to you 😊. Except all the pics above in the this document, I have added a more detailed documentation with framework, engine, feature, dev log, screen shots and demo videos through my development period onto a github repo.  Here is the link:  <https://github.com/Z-qie/Game-Cpp-Ziiky>  (Expects 2-3 mark.) |
| 0-1 | **18. Additional complexity, pre-agreed in advance with Jason (max 1 mark).**  In general this is for advanced things which don’t fit other criteria. | **CaveGenerator tilemanger class**    This is a rouge-like survival game. Therefore, I made an **auto-generated** cave map with size 12800 \* 7200 which the player can explore in the cave with a 720 \* 720 sized camera following.   1. Implemented this by first randomly by given **seed** setting each tile type of the map as cave tile/empty ground tile (please see randomFillMap() in **CaveGenerator** ) and then using a **cellular-automata** **algorithm** (please see smoothMap() in **CaveGenerator** ) to **smooth** the noised map several times(based on **strength**) to **converge** all same tiles to generate several caves. 2. After that, I implemented a **flood-fill algorithm** (please see refineMap() in **CaveGenerator )** to get all caves I got and refine the map by eliminating the cave with invalid size (too small or too big). Finally, I implemented another modified **TSP AI algorithm** (please see connectClosestRooms() in **CaveGenerator)** to connect all caves by finding the shortest path btw two tiles that are edges of every two caves and creating paths btw them by setting all tiles whose coordinates are overlapped by the lines btw these pairs of edges with a pre-set width (please see createPassage()). 3. When connecting caves, by making sure all caves are connected, I set the cave with the largest size as main cave and make each cave is either connected to the main cave or the cave it connected with is connected to the main room and recursively. Therefore, all caves are achievable by the player after this process. 4. During the generation, this generator will also initialize other tile types such as rune's position, dead walls, enemy spawning points and player's initial position.   (Expects 1 mark) |
|  |  |  |
|  | Your program crashes on exit or has a clear memory leak. (Lose 10% of your mark.) | |
|  | Your program crashes at least once during its operation. (Lose 20% of your mark.) | |
|  | Your program crashes multiple times. (Lose 30% of your mark.) | |
|  | Your program crashes frequently. (Lose 40% of your mark.) | |
|  | Your program has some odd/unexpected behaviour/errors. (Lose 10% of your mark.) | |
|  | Your program has a lot of unexpected behaviour/errors. (Lose 20% of your mark.) | |
|  | Your program crashes on exit or has a clear memory leak. (Lose 10% of your mark.) | |

# Further documentation section:

## Complex intelligence on an automated moving object

**Enemy: Gallo**

## 1. We have two types of enemies. Gallo, the first one, is a subclass of LivingEntity which can be killed by the player.

## 2. **States**: gallo has 5 states and 4 behaviors:

## 1. **Inactive**: all **dead/asleep** gallos are inactive which are **invisible** and to be re-spawned by **EnemySpawner at a position out of the view (see EenemySpawner below).**

## 2. **Wander**: when gallos are far enough but not too far (300-1000 radius) from the player, they just keep **wandering** in a random speed and to a random position (100 radius from current position **in an updating frequency as 5s max** (i.e if achieved earlier, then update directly to another random position, if not achieved in 5s, update forcefully to change position) while **keep detecting the player**. By doing this in 100 radius offset, if a player kills all the enemies around and stand still at the same position, the possibility that an enemy just being spawned can detect the player will be lower than if the player keeps moving.

## 3. **Chase**: when detecting the player, the gallo starts to chase the player with speed a little faster than player. **The target position is updated in a pre-set frequency (1s).**

## 4. **Attack**: when the gallo is **close enough (50 radius)** to the player, it starts to attack the player in a frequency of 0.3s with damage 30 per attack (see animation below).

## 5. **Sleep**: when the gallo is even farther (1000 radius) from the player and consider the player will not be approached in the recent future, it **stops wandering** and makes itself sleep and waits for the **EnemySpawner** to awake it at a new position again near the player.

## 6. **OnDeath**: when killed by the player, the gallo starts to play **the death effect** before finally disappearing. (see animation below)

**EnemySpawner**

All gallos are controlled by EnemySpawner.

1. the max number of gallos alive in the game: 25 max, start from 0 to slowly teach the player.

2. the interval of spawning is 0.8s.

3. the initial position of each gallo spawned: 400 radius from player(outside the camera's view), to give the player **a feeling that the enemies have been already there.**

**Gallo's Animation**

1. All animations of all states of gallo are made by code (procedural animation) without images.

2. **Wander & Chase**: When wandering, the colour of eyes of gallos are black with fuzzy feathers (by drawForegroundLine and oval.hollowOval…); when detecting and chasing the player, they turn to red.

3. **Attack**: When attacking the player, they **transform** to a new attacking formation:

1. growing its size at **a linear interpolated(smoothed)** speed till max radius;

2. using their dark feathers to attack the player.

4. **Death**: On death, they transform to a new dying formation:

1. turning into white colour;

2. growing its size at a **linear interpolated** speed while shrink/flash their radius by **a ping-pong algorithm** till vanishing.

5. They can turn left and right and display themselves correctly based on left and right direction.

6. All shadowing effects are coded by foregroundhollowOval and Oval

(See **Gallo, GalloProceduralAnimator and EnemySpawner** in **EnemySystem** folder to check more details. I didn’t use advanced path finding algorithm because 1. Too many enemies using this kind of algorithm will drop the performance significantly; 2. The game is a fast real-time survival rouge-like game, a fully precise path finding cant help for this as any living enemy will be dead soon when encountering the player)

## Impact/impression/wow factor:

I have added a more detailed documentation as markdown file (I think it would be more easy for you to read) with **framework**, **engine**, **features, dev log**, **screen shots and demo videos** through my development period onto a github repo.

Here is the link:

<https://github.com/Z-qie/Game-Cpp-Ziiky>

(Don’t forget to include video if you are going for 3 marks ‘sellable quality’ OR you think it MAY meet those criteria.)

­­