Assignment Guidance and Front Sheet

This front sheet for assignments is designed to contain the brief, the submission instructions, and the actual student submission for any WMG assignment. As a result the sheet is completed by several people over time, and is therefore split up into sections explaining who completes what information and when. Yellow highlighted text indicates examples or further explanation of what is requested, and the highlight and instructions should be removed as you populate ‘your’ section.

**To be completed by the student(s) prior to final submission:**

Your actual submission should be written at the end of this cover sheet file, or attached with the cover sheet at the front if drafted in a separate file, program or application.

|  |  |
| --- | --- |
| **Student ID or IDs for group work** | **2064320** |

**To be completed (highlighted parts only) by the programme administration after approval and prior to issuing of the assessment; to be consulted by the student(s) so that you know how and when to submit:**

|  |  |
| --- | --- |
| **Date set** | October 2024 |
| **Submission date (excluding extensions)** | Monday 11th November 2024 by 12:00pm (UK time) |
| **Submission guidance** | To be submitted electronically via Tabula |
| **Late submission policy** | If work is submitted late, penalties will be applied at the rate of **5 marks per University working day** after the due date, up to a **maximum of 10 working days** late. After this period the mark for the work will be reduced to 0 (which is the maximum penalty). “Late” means **after the submission deadline time as well as the date** – work submitted after the given time even on the same day is counted as 1 day late.  For **Postgraduate** students only, who started their **current course before 1 August 2019**, the daily penalty is **3 marks** rather than 5. |
| **Resubmission policy** | If you fail this assignment or module, please be aware that the University allows students to remedy such failure (within certain limits). Decisions to authorise such resubmissions are made by Exam Boards. Normally these will be issued at specific times of the year, depending on your programme of study. More information can be found from your programme office if you are concerned. |
|  |  |

**To be completed by the module owner/tutor prior to approval and issuing of the assessment; to be consulted by the student(s) so that you understand the assignment brief, its context within the module, and any specific criteria and advice from the tutor:**

Post Module Assignment Submission form

|  |  |
| --- | --- |
| **Module title &code** | WM908-15 Programming and Fundamental Algorithms |
| **Module owner** | Kurt Debattista |
| **Module tutor** | Kurt Debattista |
| **Assessment type** | Programming assignment and report |
| **Weighting of mark** | 100% |

|  |  |
| --- | --- |
| **Word count** | 3200 words equivalent for the report and code |
| **Module learning outcomes (numbered)** | Master a sound, conceptual understanding of the theory and concepts of programming and fundamental algorithms and data structures.  Autonomously distinguish the right solution for a given problem from amongst a set of algorithmic and programming tools.  Program effectively and independently in a high-level programming language at an intermediate level.  Take, straightforward to complex, algorithmic concepts, whether created or based on literature and implement them correctly. |
| **Learning outcomes assessed in this assessment (numbered)** | Master a sound, conceptual understanding of the theory and concepts of programming and fundamental algorithms and data structures.  Autonomously distinguish the right solution for a given problem from amongst a set of algorithmic and programming tools.  Program effectively and independently in a high-level programming language at an intermediate level.  Take, straightforward to complex, algorithmic concepts, whether created or based on literature and implement them correctly. |
| **Marking guidelines** | Please see Overview section. |
| **Academic guidance resources** | Reading list, module slides, standards and high-quality research and technical papers |

**The following is pre-populated for PGT assignments only:**

|  |
| --- |
| **Writing your Post-Module Assignment (PMA): specific additional advice for WMG’s Postgraduate Taught Students** |
| As a postgraduate level student in WMG you may have some concerns about your ability to write at the high standard required. This short guide is intended to provide general guidance and advice. It is important that if you have any questions you discuss them with your module tutor. Remember, in writing your PMA you need to meet the expectations of the reader and university. |
| **A good PMA generally requires you to answer the question and to include**…   1. A title, with your student number, module, lecturer’s name and any other documentation required by the university. 2. A contents page and if appropriate, an abstract. 3. An introduction which acts as a ‘map’ to the rest of the document, describing the aim or purpose of the work and explaining how this aim is achieved. At this point it is usually helpful to paraphrase your conclusion. 4. Evidence of an appropriate level of background reading of relevant texts. 5. Evidence of systematic and clear thinking, indicative of good planning and organisation. 6. Writing which makes sense, is clearly and carefully presented (proof-read and grammar checked). 7. A critical style of writing which compares and contrasts the main theories, concepts and arguments with conclusions that are based in evidence presented. 8. High levels of accurate academic referencing. 9. A logical and well-defined structure with headings and subheadings. 10. Clearly labelled and well-presented diagrams and other graphics that are discussed in the text. 11. Adherence to usual academic standards including length and a timely submission. 12. A reference section in which every source that is cited in the text is listed. |
| **Where to get help:**   1. **Talk to your module tutor if you don’t understand the question or are unsure as to exactly what is required.** 2. Study, Professional and Analytical Skills (SPA) Moodle site – we have a lot of resources on this website with workbooks, links and other helpful tools. <https://moodle.warwick.ac.uk/> 3. The university Academic Writing centre provides workshops and useful tools to help you in all aspects of your work. <https://warwick.ac.uk/services/skills/academicwriting/> 4. Avoiding Plagiarism, the university’s site to help you to reference properly <https://moodle.warwick.ac.uk/course/view.php?id=42224> 5. Wellbeing support services <https://warwick.ac.uk/services/wss> 6. Numerous online courses provided by the University library to help in academic referencing, writing, avoiding plagiarism and a number of other useful resources. <https://warwick.ac.uk/services/library/students/your-library-online/> |

Assignment Brief

This assignment is to develop a small 2D game using the provided ***GamesEngineeringBase*** framework. This framework will be presented in an in-class tutorial. The program needs to be developed in C++, and the code submitted with a report, details of which are outlined below. An executable should also be submitted.

Please read the entirety of the assignment before attempting any part of it, as thinking about the overall solution from the start will be beneficial for some of the later stages.

Show your code for all parts, even if you do not get the final result. If you cannot provide code, at least explain your reasoning around the problem and how you would solve it. Marks will be given for both code and explanations – more marks will be allocated for proper code that works. If you cannot provide the full functionality at any stage, provide part of it but state which part was provided.

Provide comments in the code – the more readable it is the easier it is to understand and allocate marks. Please include Visual Studio solutions and all the source code, or a link to a GitHub Repository. If you wish to use another compiler and IDE you are permitted to, but please contact the module tutor before you do so, such that arrangements for marking are made in a timely fashion.

Make sure you do not copy any code from the internet or online resources or from any other students. This includes the use of AI to generate code.

Use of the Standard Template Library data structures (such as vector, list, map) is not permitted for this assignment. No external APIs can be used either. You are allowed to use and build upon code we developed in class including any tutor provided solutions.

This assessment will help you re-enforce the learning from the module. You will need to understand how to use C++ to develop games and provide solutions to be used directly as part of the game.

# Overview

This section provides an overview of the game and the components which should be included. The game is a 2D survivor like game (like Vampire Survivors). The game logic is expected to be relatively simple, and to be built around a core **game loop** with simple updating of the player, **game world**, and Non-Player Characters (NPCs).

When designing solutions think of the efficiency of the solution and which data structures and algorithms best help support the solution. Ensure these are discussed in the report with a discussion on complexity.

# Game (85 Marks)

This section carries a total of 85 marks. However, the individual aspect of the game is evaluated on the report of it (see Section 3 Report).

This game should be built on the core game systems developed during tutorial sessions and must include implementations of the following:

1. A virtual camera that follows the player-controlled (hero) character at its centre (**10 Marks**)
2. A number of NPCs that attack the character (**17 Marks**)
   * + Generated randomly outside of camera view (2 Marks)
     + Their frequency increases over time (2 Marks)
     + At least 4 different character types that differ in appearance, health and speed (5 Marks)
     + General NPC behaviour that directs them directly towards the player (4 Marks)
     + One NPC behaviour makes it static but launches projectiles (4 Marks)
3. Collision system (**12 Marks**)
   * + Hero vs NPCs (3 Marks)
     + Hero vs impassable terrain (3 Marks)
     + Hero projectiles vs NPCs (3 Marks)
     + NPC projectiles vs Hero (3 Marks)
   * Clearly demonstrate how NPCs are implemented and handled
4. The hero attacks the NPCs with at least two different types of attack (**12 Marks**)
   * A linear attack that targets closest NPC (has cooldown) – the attack runs automatically all the time (3 Marks)
   * A special area of effect (AOE) attack that targets the top N max health NPC – triggerred by hero via separate button (can be instantanous but has significant cooldown). (7 Marks)
   * A powerup that increases either the speed of the linear attack and number N of NPCs targetted by the AOE (2 Marks)
5. A tile-based method (composed of a number of 32x32 pixel tiles) for displaying the background (**16 Marks**)
   * At least four different tile types (ie terrain types). One of which is impassable (eg water) for the hero. NPCs can traverse. (5 Marks)
   * Data driven level loading, i.e. load the game world’s tiles and map from a custom file format (5 Marks)
   * A version of the world which is infinite (with repeating tiles) (3 Marks)
   * A version with a fixed boundary (larger than what the screen displays) (3 Marks)
6. Game level runs for two minutes (**7 Marks**)
   * Show at least two levels with different maps (one infinite, one fixed) (5 Marks)
   * Score and FPS shown at the end (or during) (2 Marks)
7. The ability to save the current state of the game (and save to a file) at any point and reload it at the same exact point (**11 Marks**)

Please note that we will not be marking the game, rather we will mark the implementations of the above methods. Please use the report to explain your different solutions. You can show screenshots abstracted from the gameplay e.g. an image showing your collision detection system.

# Report (15 marks)

The report should be structured as follows:

* Introduction which introduces your game, and the technologies used
* A section on each of the technologies implemented (1-6 from Section 2) which contains details about how the technique works and how you implemented it. Half of this section should cover theory; half should cover the relevant implementation details for each method. You should include details here even if your implementation is partially complete.
* A short evaluation section – measure the FPS / ms per frame and see how this varies as the level complexity increases
* Limitations which cover the things you tried that did not work or if there are bugs that you know about but could not fix
* Conclusion which summarises the report in a single paragraph

Also add to the report a small section on how you would have approached this project differently if you had to start from scratch after this learning experience.

You are expected to include screenshots of the game running and implementations of game systems. For this you can use an application such as the “snipping tool” in Windows to directly paste the results in your document – this is very quick. Windows 10 has a new screen capture facility using Shift+MSkey+S. Similarly, most of the latest versions of Mac OSX permit the use of screen capture via shift-cmd-4 (various numbers provide different functionality).

Do not add the code to the appendix but provide it separately and in a format such that it can be compiled directly (see Section 1).

**Important:** All samples of code shown in the text need to be in text format, **not** a screen capture from your editor. Code in screen captures **will be ignored**.

**Report**

Student number: u2064320

Submission date: 2024.11.10

Github link of the code: [AmbmbmA/WM908-Assignment: WM908 Assignment, u2064320](https://github.com/AmbmbmA/WM908-Assignment)

**Introduction of the game**

The game is coded within one single file. The executable **requires 3 folders to work** normally. **“Resources” folder** is for games art resources, most of the arts are cc0 resources from the website ” opengameart.org”, the rest of them all drawn on my own. **“Save” folder** is for an initial save data to start a new game, and also for the saved game data and random map seed. **“World” folder** has some predefined finite map seed to be load.

The game has an **UI hand written** by myself, the interact logic is to detect the coordinate of the mouse to get which button is chosen, position is predefined based on how I drawn them.

The game has **2 levels with 2 save and load slot**. The load function works even you restart the game file.

There is brief game control method in the game control option. There is also **an readme file** explained the detail of how to control the player. The in game data such as fps and level up, is shown on the console. There would be an **conclusion of the latest game you played on console** once you quit the game. Rest of the function are build in UI, you could just click to use them.

卡通人物

描述已自动生成地图

描述已自动生成

**Implementation**

**Brief Structure and design**

The initial part contains **game constant** I modified frequently to change the game play design. Such as the attack value, speed etc.

Then there is one **template** for **double linked list**, I use it store NPCs and Projectiles.

**Classes**

I have a **base class** called **Sprites** to do basic image loading and drawing(with bound detect).

**Player class** is **inherits from Sprites.** It has methods for level up, update its status, save and load player data.

**NPC class** is also **inherits from Sprites.** It has methods to update its status and to save one NPC data.

**Spawn class** is a class that has a **double linked list of NPC class’s objects’ pointers**. It has methods to control the **probability of each type of NPCs** to be spawned, to **generate** NPC, to detect **collision between NPCs and Player**, to **check** weather a NPC is needed to be **removed**, to update and draw all Spawned, to save the Spawn data, to load spawn data and **load data for each NPC** in the double linked list.

**Projectile class** is **inherits from Sprites.** It has methods to update its status and to save one projectile data.

**Projectilemanage class** is a class that has two **double linked lists of Projectile class’s objects’ pointers**.

One is for Player, one is for NPC. Each of them have methods to **generate and check deletion**. It has similar structure with the Spawn class.

**AOE class** has methods to **check if a NPC is within the targeted area**, to **detect which of the NPCs is being targeted,** to draw attack visual effect, aim and target effect, to **attack, update cooling down and power up** information, and for save and load.

**Tile class** is basically a sprites class.( Could be improved as one)

**Tileset class** has an **dynamic array of tiles’ objects’ pointers**. It loads all the tile files.

**World class** is in charge of world map based on an **dynamic 2d array as map seed**. It has methods to **create map randomly based on a certain probability** of different tile types, to **load map seed from file**, to change map seed after initializing, to **save map seed to file**, to draw the map, to **collides with the player**.

**UI class** has methods **to draw player’s health**, detect for different UI which button is being clicked.

The **save and load functions** are based on those within then classes, and some global data.

The **main function,** creates objects of classes, draw the canvas. A **game loop to control the game run**.

Within it there are two branch, one is **actual game running with level loop**, the other is **pause the game to UI**. After game loop are some **end game display**.

**1.Centre camera**

The canvas is the camera, in my code, the player is **always drawn at the center of the canvas**. While all the move of the player is **actually moving the map**, so when the player tries to move, it update the wx and wy for the map to be drawn. I also have a wxp and wyp to monitor the position of the player with respect to the map in case I want some further function, but for this case it is not used.

**-------------------------------------------------------------------------------------------------------------------------------------**

Player p("Resources/Player1.png", canvas.getWidth() / 2, canvas.getHeight() / 2, 0);

**-------------------------------------------------------------------------------------------------------------------------------------**

if (canvas.keyPressed('W') && canvas.keyPressed('A')) { //left above

dy -= speed \* 0.01 \* u \* sqrt(2) / 2;

dx -= speed \* 0.01 \* u \* sqrt(2) / 2;

}

else if (canvas.keyPressed('W')) { //above

dy -= speed \* 0.01 \* u;

}

if (dx >= 3) {

wx += 3;

wxp += 3;

dx = 0;

}

**--------------------------------------------------------------------------------------------------------------------------------**

For all movement update, I use something similar to this. So the form each frame to update could be **a float instead of int value**, so it would be more accurate and I can have more sensitive speed change. For the player, I add the check if it is trying to **move in a diagonal direction**, then change the dy dx update value **from one unit to sqrt(2) / 2**, so I would move faster than non-diagonal cases.

**2.NPCs**

To control **the generate area which is a rectangular ring** between the INMARGIN outside of the canvas and OUTMARGIN outside of the canvas(thus the camera). We first random X with a value between -OUTMAGIN to canvas width + OUTMARGIN, if the value is outside of the two INMARGIN, then Y could be random between -OUTMAGIN to canvas height + OUTMARGIN. Otherwise I use “rand() % 2 “ to decide above or below equally, then give Y a random value in proper range.

**--------------------------------------------------------------------------------------------------------------------------------**

//random position

int randomX, randomY;

randomX = rand() % (canvas.getWidth() + 2 \* OUTMARGIN) - OUTMARGIN;

// X [-OUTMARGIN,width+OUTMAGIN]

if (randomX < -INMARGIN || randomX > canvas.getWidth() + INMARGIN) {

//x ouside INMARGIN

randomY = rand() % (canvas.getHeight() + 2 \* OUTMARGIN) - OUTMARGIN;

// y [-OUTMARGIN,height+OUTMAGIN]

}

else { // x inside canvas

if (rand() % 2 == 0) { //50% above

randomY = -INMARGIN - rand() % (OUTMARGIN - INMARGIN);

//y [-INMARGIN,-OUTMARGIN]

}

else { //50% below

randomY = canvas.getHeight() + INMARGIN + rand() % (OUTMARGIN - INMARGIN);//y [height + INMARGIN,height + OUTMARGIN]

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

Below is to make sure the **maximum number of NPC allowed to exist**, is it is full, it would skip the generate part of the code. Also the NPC would **only be spawned after time has passed a certain value**, this **value would be lower each time until it meets the lower bond** set for that level ( **once passed** **reset back, then skip this test for following times**), the **time pass is monitored by adding dt value** for each frame.

**--------------------------------------------------------------------------------------------------------------------------------**

void generate(Window& canvas, Player& p, int wx, int wy, float dt, int level) {

timeElapsed += dt;

bool full = false;

if (npc.getsize() >= MAXNUM) {

full = true;

}

if (timeElapsed >= timeThreshold && (!full)) {

//random position

// random npc

timeElapsed = 0.0f; //reset

if (timeThreshold != MINSPAWNGAP[level]) {

// once reach limit, do not change

timeThreshold -= SPAWNACC; // accelerate spawn rate

if (timeThreshold <= MINSPAWNGAP[level]) {

timeThreshold = MINSPAWNGAP[level];

// restrict the min gap

}

}

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

If all the above conditions are true, then we start to generate NPCs, by load a file, the file is chosen randomly with certain probability to each type of NPCs, then use that file to **initialize a NPC object and add the object to a double linked list**. Using double linked list is because I want it to have a **dynamic size**, also I want to monitor the size, **delete some elements in the middle**, so this allows me **optimize the memory** used for Spawning.

**--------------------------------------------------------------------------------------------------------------------------------**

// random npc

int npcindex = randomnpcindex();

string filename = "Resources/npc" + to\_string(npcindex) + ".png";

//create npc

NPC\* n = new NPC(filename, randomX, randomY, wx, wy, npcindex);

generated[npcindex]++;

npc.addend(n);

**--------------------------------------------------------------------------------------------------------------------------------**

The character type is control by npcindex which is random based on certain probabilities when generating, they are **assigned with different resource files and different speed and health from the constant arrays in the beginning.** The static one has the speed of zero.

**--------------------------------------------------------------------------------------------------------------------------------**

//health

const unsigned int PLAYERMAXHEALTH[1] = { 3000 }; // player health

const unsigned int NPCMAXHEALTH[4] = { 2000 , 3000 , 4000 , 5000 }; // npc health

//speed

const float PLAYERSPEED[1] = { 60 }; // player speed

const float NPCSPEED[4] = { 50 , 40 , 30 , 0 }; // npc speed

**--------------------------------------------------------------------------------------------------------------------------------**

Then when update them, **all of the NPC would move based on the world move**( which is in fact , player move), **this shows the relative move between all NPCs and the player.**

Then if it is **not the Static one**, we give set them with aiming to the player position, it **calculate the length between them to try to move until they meet**. The ux and uy are **calculated by vector direction** so it moves in straight line.

**--------------------------------------------------------------------------------------------------------------------------------**

void update(Window& canvas, Player& p, int wx, int wy, float u) {

// update xy based on the change of world

x += wxi - wx - cx;

y += wyi - wy - cy;

cx = wxi - wx;

cy = wyi - wy;

int px = p.getcX(); int py = p.getcY(); // player world position

int difx = px - cx;

int dify = py - cy;

length = sqrtf(difx \* difx + dify \* dify);

if (npcindex != 3) { // skip static one

// always towards player

float ux = 0.0f; float uy = 0.0f; // direction scaler

if (length >= (p.getsprite().width + getsprite().width) / 2) {

ux = difx / length;

uy = dify / length;

}

float \_dx = speed \* 0.01f \* ux \* u;

float \_dy = speed \* 0.01f \* uy \* u;

dx += \_dx;

dy += \_dy;

}

**--------------------------------------------------------------------------------------------------------------------------------**

Then for the static one to shoot projectiles to the player. The **update function of Projectile class only needs the targeted coordinates**. All the rest **is similar to the NPC update function**. I create an double linked list for it in the projectilemanage class, I also let **it delete if it has traveled for a certain distance(mesure by mt) ,** so it won’t chase the player forever, I did that for the player projectiles also but with a different length.

**--------------------------------------------------------------------------------------------------------------------------------**

Void update(Window& canvas, int arget, int arget, int wx, int wy, float u) {

// update xy based on the change of world

x += wxpr – wx – cx ;

y += wypr – wy – cy ;

mt += abs(wxpr – wx – cx) + abs(wypr – wy – cy) ;

cx = wxpr – wx;

cy = wypr – wy;

int difx = arget – cx;

int dify = arget – cy;

length = sqrtf(difx \* difx + dify \* dify);

// always towards target

float ux = 0.0f; float uy = 0.0f; // direction scaler

if (length >= 0) {

ux = difx / length;

uy = dify / length;

}

float \_dx = speed \* 0.01f \* ux \* u;

float \_dy = speed \* 0.01f \* uy \* u;

dx += \_dx;

dy += \_dy;

mt += abs(\_dx) + abs(\_dy);

**--------------------------------------------------------------------------------------------------------------------------------**

**3.Collision**

**Spawn class has a method pvn** detects weather the player and each NPC in the double linked list has a **distance too close (set as half of the sum of their width) to each other**. In that case, each frame, both of them would be **receive a crash damage** of 1, I set all health to be relatively high so even it is updating by frame, it won’t be too quick. Separate next is used **because in case the NPC dies, the node would be deleted, then it would not be available.**

**--------------------------------------------------------------------------------------------------------------------------------**

void pvn(Player& p) {

node<NPC\*>\* current = npc.gethead();

while (current != nullptr) {

node<NPC\*>\* next = current->next;

if (current->data->length <= (p.getsprite().width + current->data->getsprite().width) / 2) {

p.health -= CRASH;

current->data->health -= CRASH;

}

current = next;

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

World class has a method collisionplayer, it **go through map seed and only detect for those tile with a index** **that has collision ability**, then **based on the position of the player relative to the tile**, it would detect weather there is a collision and if there is, it gives an opposite update of wx and wy with same speed to player, so it **cancel the movement**. I tried to add alpha detect to make it pixel accurate, but it somehow doesn’t work, I would still collides with the whole image rather than the part drawn.

**--------------------------------------------------------------------------------------------------------------------------------**

if (mapseed[currentX][currentY] > 4) {

// for all position where has drawn pixel of player

for (int ii = p.getX(); ii < p.getX() + p.getsprite().width; ii++) {

for (int jj = p.getY(); jj < p.getY() + p.getsprite().height; jj++) {

// check collision

if (ii >= drawX && ii < drawX + tilewidth && jj >= drawY && jj < drawY + tileheight) {

col = true;

bool left = (ii == p.getX());

bool right = (ii == p.getX() + p.getsprite().width - 1);

bool up = (jj == p.getY());

bool down = (jj == p.getY() + p.getsprite().height - 1);

if (left && up) { //left above

p.dy += p.speed \* 0.01 \* u \* sqrt(2) / 2;

p.dx += p.speed \* 0.01 \* u \* sqrt(2) / 2;

}

The two projectiles are basically the same as the NPC vs Player one, except **the generate position is at the NPC or the Player who shoot it**, and target coordinates. For the NPC one, it behaves like an NPC **spawned but delete itself once collision happened.**

**--------------------------------------------------------------------------------------------------------------------------------**

void generate0(Window& canvas, Player& p, int wx, int wy, float dt) {

timeElapsed0 += dt;

if (timeElapsed0 > p.shootgap) { // player shoot gap

string filename = "Resources/playerpro.png";

//create proj

Projectile\* projn = new Projectile(filename, p.getcX(), p.getcY(), wx, wy, 0);

proj0.addend(projn);

timeElapsed0 = 0.0f; //reset

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

While the Player one has **extra detect that go though NPC double linked list to choose the one with lowest distance** as the target.

**--------------------------------------------------------------------------------------------------------------------------------**

// proj0

node<NPC\*>\* currentn = s.npc.gethead();

if (currentn != nullptr) { // check if there is npc

generate0(canvas, p, wx, wy, dt);

node<NPC\*>\* target = s.npc.gethead(); // closesr to player

//go through all npc

while (currentn != nullptr) {

if (currentn->data->length < target->data->length) {

target = currentn;

}

currentn = currentn->next;

}

// go through each proj0 in the list

node<Projectile\*>\* currentp0 = proj0.gethead();

while (currentp0 != nullptr) {

node<Projectile\*>\* next = currentp0->next;

int targetx = target->data->getcX();

int targety = target->data->getcY();

currentp0->data->update(canvas, targetx, targety, wx, wy, u);

checkdelete0(canvas, currentp0, target);

currentp0 = next;

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

void checkdelete0(Window& canvas, node<Projectile\*>\* proj, node<NPC\*>\* npc) {

int rightb = canvas.getWidth() + OUTMARGIN;

int leftb = -OUTMARGIN;

int bottomb = canvas.getHeight() + OUTMARGIN;

int upb = -OUTMARGIN;

if (proj->data->getcX() > rightb ||

proj->data->getcX() < leftb ||

proj->data->getcY() > bottomb ||

proj->data->getcY() < upb) {

proj0.remove(proj);

}

else if (proj->data->length <= 10) {

proj0.remove(proj);

npc->data->health -= PROJDAMAGE[0];

}

else if (proj->data->mt >= PROJMAXT[0]) {

proj0.remove(proj);

}

else if (npc == nullptr) {

proj0.remove(proj);

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

**4.Hero**

The hero linear attack is similar to NPC spawn, **it generate a projectile at the center of the NPC position**, then as mentioned above, it set the closest NPC as target to move, then delete itself once the collision happened.

The special AOE has a **specialized class** to do so. It **is targeting an circular area based on the mouse position**. I use a method to draw a circular ring with green cross at a given coordinate. A method id for **detecting weather NPCs is within the area**, by **checking its distance to center is larger than the radius or not.**

bool checkwithin(node<NPC\*>\* n, int aoex, int aoey) {

float difx = n->data->cx - aoex;

float dify = n->data->cy - aoey;

float distance = sqrtf(difx \* difx + dify \* dify);

if (distance <= aoer) {

return true;

}

return false;

}

**--------------------------------------------------------------------------------------------------------------------------------**

图片包含 背景图案

描述已自动生成地图

描述已自动生成

I then use a method to **go through all NPC, for those within the area, I compare there health to put the highest 4 into a dynamic array as target**( so I could increase the size later). I delete the array each frame and recheck to make it accurate.

**--------------------------------------------------------------------------------------------------------------------------------**

void detect(Spawn& s, int aoex, int aoey) {

if (aoetarget != nullptr) {

delete[]aoetarget;

}

aoetarget = new NPC \* [aoenum];

for (int i = 0; i < aoenum; i++) {

aoetarget[i] = nullptr;

}

node<NPC\*>\* current = s.npc.gethead();

while (current != nullptr) { // for all npc

node<NPC\*>\* next = current->next;

if (checkwithin(current, aoex, aoey)) { // check within the aoe area

bool added = false;

for (int i = 0; i < aoenum; i++) {

if (aoetarget[i] == nullptr) {

aoetarget[i] = current->data;

added = true;

break;

}

}

if (!added) {

int lowest = 0;

for (int i = 1; i < aoenum; i++) {

if (aoetarget[i]->health < aoetarget[lowest]->health) {

lowest = i;

}

}

if (current->data->health > aoetarget[lowest]->health) {

aoetarget[lowest] = current->data;

}

}

}

current = next;

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

When aiming, it draw the aim effect **and if the cooling down is ready draw smaller version at those NPC in the targeted array.**

**--------------------------------------------------------------------------------------------------------------------------------**

void draw(Window& canvas, Spawn& s, int aoex, int aoey) {

drawaim(canvas, aoex, aoey, aoer);

detect(s, aoex, aoey);

if (!cooling) {

for (int i = 0; i < aoenum; i++) {

if (aoetarget[i] != nullptr) {

drawaim(canvas, aoetarget[i]->cx, aoetarget[i]->cy, aoetarget[i]->getsprite().width / 2);

}

}

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

As to attack and cool down. I have one method to draw the attack effect as a black circle. **Then bool variable cool controls whether it is cooling. An update method to monitor if it is ready.**

**--------------------------------------------------------------------------------------------------------------------------------**

void update(Player& p, float gametime) {

if (cooling) {//check cd

if (gametime - lastatk <= 0 || gametime - lastatk >= cd) {

cooling = false;

}

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

A atk method to do the actually attack when it is ready.

**--------------------------------------------------------------------------------------------------------------------------------**

bool atk(Window& canvas, Spawn& s, int aoex, int aoey, float gametime) {

if (!cooling) {

detect(s, aoex, aoey);

for (int i = 0; i < aoenum; i++) {

if (aoetarget[i] != nullptr) {

aoetarget[i]->health -= AOEDAMAGE;

}

}

lastatk = gametime;

cooling = true;

return true;

}

else {

return false;

}

}

The **power up is based on the level up system**. The player has 11 levels (0-10), level up is based on the score the player get. It **gets more as higher level you reach**. **Once level up, the bool variable power up is set as true, once power up effect is updated, it resets back to false waiting for next level up.**

**--------------------------------------------------------------------------------------------------------------------------------**

int scoretolevelup() {

int scoretoup = 0;

for (int i = 0; i <= plevel; i++) {

scoretoup += BASESCOREFORUP \* (i + 1);

}

return scoretoup;

}

For odd levels, **player shooting gap is shorter**

if (plevel < MAXPLEVEL && score >= scoretolevelup()) {

cout << "Score reach " << scoretolevelup() << endl;

plevel += 1;

powerup = true;

cout << "PLAYER LEVEL UP --> " << plevel << endl;

}

if (plevel % 2 != 0 && powerup) { // powerup when odd level

shootgap -= 0.06;

cout << "SHOOT SPEED UP!" << endl;

powerup = false;

}

**--------------------------------------------------------------------------------------------------------------------------------**

For even levels, **the radius of the AOE circle and max number as targeted is increased**

**--------------------------------------------------------------------------------------------------------------------------------**

if (p.plevel % 2 == 0 && p.plevel != 0 && p.powerup) { // powerup when even level

aoenum += 1;

aoer += 20;

cout << "AOE ENHANCED!" << endl;

p.powerup = false;

}

**--------------------------------------------------------------------------------------------------------------------------------**

**5.Tile and map**

Tile class is similar to Sprites class, it loads the file of tiles, it also has the ability to get its size.

Then the tileset loads all the files for the game, **there are 9 tiles, 4 of them are impassable. The method is explained in the collision part.**

The world is **drawn based on an dynamic 2d array called mapseed**. It contains how the map is lay out using the tiles.

电脑萤幕画面

低可信度描述已自动生成 图形用户界面, 应用程序

描述已自动生成

The **infinity of the world is obtained by % the size of the world when drawing**, so **it repeat itself** in all directions.

**--------------------------------------------------------------------------------------------------------------------------------**

void draw(Window& canvas, int wx, int wy) {

int tilewidth = tiles[0].getwidth(); // get the standard width for this tileset

int tileheight = tiles[0].getheight(); // get the standard height for this tileset

int nw = canvas.getWidth() / tilewidth; //number of tiles can be put in width

int nh = canvas.getHeight() / tileheight; //number of tiles can be put in height

int X = wx / tilewidth; // which tile wx is on

int Y = wy / tileheight; // which tile wy is on

int offsetx = wx % tilewidth;

int offsety = wy % tileheight;

for (int i = -1; i < nw + 1; i++) { // [-1,1] so there is no black gap when it touch the edge

for (int j = -1; j < nh + 1; j++) {

int currentX = (X + i) % worldsizeX;//which tiles to be draw, mod the maxsize to make it loop

int currentY = (Y + j) % worldsizeX;

int drawX = i \* tilewidth - offsetx; // first draw consider the firsr tile with offset value, then add the rest based on where it is on the axis

int drawY = j \* tileheight - offsety;

tiles[mapseed[currentX][currentY]].draw(canvas, drawX, drawY);

}

}

}

The mapseed could be random based on a certain probability. If want finite world, even for random map, it would **build a ring of impassable tiles around the map**, the thick of it is greater than half of the length and width of the canvas, so it is drawed properly since the player stays in the middle.

**--------------------------------------------------------------------------------------------------------------------------------**

world(unsigned int \_worldsizeX, unsigned int \_worldsizeY, int \_level, string \_tiletype = "t") {

if (LEVELMAPINF[\_level] == true) {

worldsizeX = \_worldsizeX;

worldsizeY = \_worldsizeY;

tiletype = \_tiletype;

mapseed = new unsigned int\* [worldsizeX];

for (unsigned int i = 0; i < worldsizeX; i++) {

mapseed[i] = new unsigned int[worldsizeY];

}

tiles.load(tiletype);

for (unsigned int i = 0; i < worldsizeX; i++) {

for (unsigned int j = 0; j < worldsizeY; j++) {

mapseed[i][j] = randomtileindex();

}

}

}

else {

int caw = HORIBOND;

int cah = VERTIBOND;

worldsizeX = \_worldsizeX + caw \* 2;

worldsizeY = \_worldsizeY + cah \* 2;

tiletype = \_tiletype;

mapseed = new unsigned int\* [worldsizeX];

for (unsigned int i = 0; i < worldsizeX; i++) {

mapseed[i] = new unsigned int[worldsizeY];

}

tiles.load(tiletype);

// obstacle tiles

for (unsigned int i = 0; i < worldsizeX; i++) {

for (unsigned int j = 0; j < worldsizeY; j++) {

mapseed[i][j] = 5;

}

}

// world in the middle

for (unsigned int i = caw; i < (worldsizeX - caw); i++) {

for (unsigned int j = cah; j < (worldsizeY - cah); j++) {

mapseed[i][j] = randomtileindex();

}

}

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

It can also read from file, the **file specified the tile type to read the resource file, the size of the map to reset dynamic 2d array, then the tileindex to be stored in the array.**

**--------------------------------------------------------------------------------------------------------------------------------**

world(string filename) {

ifstream loadmap;

loadmap.open(filename, ios::in);

loadmap >> tiletype;

tiles.load(tiletype);

loadmap >> worldsizeX >> worldsizeY;

mapseed = new unsigned int\* [worldsizeX];

for (unsigned int i = 0; i < worldsizeX; i++) {

mapseed[i] = new unsigned int[worldsizeY];

}

for (unsigned int j = 0; j < worldsizeY; j++) { //change for loop order to give a clear view of map

for (unsigned int i = 0; i < worldsizeX; i++) {

loadmap >> mapseed[i][j];

}

}

loadmap.close();

}

**--------------------------------------------------------------------------------------------------------------------------------**

I also set the initial wx and wy so **the player begins from the middle of the map**, this mattes a lot for finite map since the map drawn form left up corner.

**--------------------------------------------------------------------------------------------------------------------------------**

//center the canvas on map

for (int i = 0; i < LEVELNUM; i++) {

wx[i] = w[i].getsizex() \* w[i].gettileset()[0].getwidth() / 2;

wy[i] = w[i].getsizey() \* w[i].gettileset()[0].getheight() / 2;

}

**--------------------------------------------------------------------------------------------------------------------------------**

**6.Game level**

Game levels is **moitered by time** **by adding dt to get the game time**, once it reaches 120s, it **reset all double linked list of NPCs and Projectiles, Game\_time for the level, and draw the new map**. **The Player status would stay the same.** To do this, all the item need to be different for each level, **I put them into arrays**. And **update gamelevel**, so the loop using the correct data.

**--------------------------------------------------------------------------------------------------------------------------------**

world w[LEVELNUM] = { w0,w1 };

// Random spawn NPC

Spawn s0, s1;

Spawn s[LEVELNUM] = { s0,s1 };

// Projectiles

Projectilemanage projl0, projl1;

Projectilemanage projl[LEVELNUM] = { projl0,projl1 };

// for in game show FPS

int framecount = 0;

float secondcount = 0.0f;

//for final FPS

int overframecount = 0;

float Game\_time[LEVELNUM] = { 0.0f,0.0f };

// world position(left up corner)

int wx[LEVELNUM] = { 0,0 };

int wy[LEVELNUM] = { 0,0 };

**--------------------------------------------------------------------------------------------------------------------------------**

The score is **updated as NPCs are defeated** as part of the player status.

**--------------------------------------------------------------------------------------------------------------------------------**

int scorenow = 0;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < LEVELNUM; j++) {

scorenow += s[j].defeated[i] \* NPCSCORE[i];

}

}

p.score = scorenow;

**--------------------------------------------------------------------------------------------------------------------------------**

The FPS to generated by **keeping a frame count and game time** to calculate the value.

**--------------------------------------------------------------------------------------------------------------------------------**

**In game**

if (secondcount >= INGAMESHOW) {

//FPS

int FPS = framecount / secondcount;

cout << "FPS:" << FPS << endl;

framecount = 0;

secondcount = 0;

}

**End game**

int FPS = overframecount / sumarr<float>(Game\_time, LEVELNUM);

if (FPS <= 0) { FPS = 0; }

cout << "Average FPS: " << FPS << endl;

**7.Save and load**

The save and load functions are separated into different classed.

For player, it **save all the position and player status to be loaded**

**--------------------------------------------------------------------------------------------------------------------------------**

void save(ofstream& save) {

save << "Player" << endl;

save << dx << endl;

save << dy << endl;

save << wxp << endl;

save << wyp << endl;

save << health << endl;

save << plevel << endl;

save << score << endl;

save << shootgap << endl;

}

void load(ifstream& load) {

string head;

load >> head;

load >> dx;

load >> dy;

load >> wxp;

load >> wyp;

load >> health;

load >> plevel;

load >> score;

load >> shootgap;

}

**--------------------------------------------------------------------------------------------------------------------------------**

NPC and Projectiles have similar save functions, but as they are **managed by Spawn or Projectilemage,** The load is build there **because we need to go through the double linked list.**

**--------------------------------------------------------------------------------------------------------------------------------**

void save(ofstream& save) {

save << "SPAWN" << endl;

save << timeElapsed << endl;

save << timeThreshold << endl;

save << npc.getsize() << endl;

for (int i = 0; i < 4; i++) {

save << generated[i] << endl;

save << defeated[i] << endl;

}

node<NPC\*>\* current = npc.gethead();

int count = 0;

while (current != nullptr) {

node<NPC\*>\* next = current->next;

current->data->save(save, count);

count++;

current = next;

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

void npcload(ifstream& load, int count) {

string head;

load >> head;

int npcindex;

load >> npcindex;

string filename = "Resources/npc" + to\_string(npcindex) + ".png";

NPC\* n = new NPC(filename, 0, 0, 0, 0, npcindex);

load >> n->dx;

load >> n->dy;

int temp;

load >> temp;

n->changex(temp);

load >> temp;

n->changey(temp);

load >> n->cx;

load >> n->cy;

load >> n->wxi;

load >> n->wyi;

load >> n->health;

npc.addend(n);

}

void load(ifstream& load) {

string head;

load >> head;

load >> timeElapsed;

load >> timeThreshold;

int size;

load >> size;

for (int i = 0; i < 4; i++) {

load >> generated[i];

load >> defeated[i];

}

if (npc.getsize() != 0) {

npc.clear();

}

for (int i = 0; i < size; i++) {

npcload(load, i);

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

AOE save is also some basic data with a bool viriable.

**--------------------------------------------------------------------------------------------------------------------------------**

void save(ofstream& save) {

int c = 0;

if (cooling) { int c = 1; }

save << "AOE" << endl;

save << aoer << endl;

save << lastatk << endl;

save << aoenum << endl;

save << cd << endl;

save << c << endl;

}

void load(ifstream& load) {

string head;

load >> head;

load >> aoer;

load >> lastatk;

load >> aoenum;

load >> cd;

int c;

load >> c;

if (c == 1) {

cooling = true;

}

else {

cooling = false;

}

}

**--------------------------------------------------------------------------------------------------------------------------------**

World is save in different ways, cause we **only need wx and wy to locate**, we only need to save **another file for the mapseed**, and ability to change mapseed after the object is created.

**--------------------------------------------------------------------------------------------------------------------------------**

void savemapseed(string filename) {

ofstream savemap;

savemap.open(filename, ios::out);

savemap << tiletype << endl;

savemap << worldsizeX << "\t" << worldsizeY << endl;

for (unsigned int j = 0; j < worldsizeY; j++) { //change for loop order to give a clear view of map

for (unsigned int i = 0; i < worldsizeX; i++) {

savemap << mapseed[i][j] << "\t";

}

savemap << endl;

}

savemap.close();

}

**--------------------------------------------------------------------------------------------------------------------------------**

void changemapseed(string filename) {

ifstream loadmap;

loadmap.open(filename, ios::in);

loadmap >> tiletype;

tiles.load(tiletype);

loadmap >> worldsizeX;

loadmap >> worldsizeY;

for (unsigned int j = 0; j < worldsizeY; j++) { //change for loop order to give a clear view of map

for (unsigned int i = 0; i < worldsizeX; i++) {

loadmap >> mapseed[i][j];

}

}

loadmap.close();

}

**--------------------------------------------------------------------------------------------------------------------------------**

Then we have a function to detect weather a file exit to be opened.

**--------------------------------------------------------------------------------------------------------------------------------**

bool fileexist(const string& filename) {

ifstream file(filename);

return file.good();

}

**--------------------------------------------------------------------------------------------------------------------------------**

Then this is the **overall save function**

**--------------------------------------------------------------------------------------------------------------------------------**

void savegame(int \_slot, int& level, float\* Game\_time, int\* wx, int\* wy, world\* w, Player& p, Spawn\* s, Projectilemanage\* proj, AOE& aoe) {

ofstream save;

save.open("Save/save" + to\_string(\_slot) + ".txt", ios::out);

save << level << endl;

w[level].savemapseed("Save/worldforsave" + to\_string(\_slot) + ".txt");

for (int i = 0; i < LEVELNUM; i++) {

save << Game\_time[i] << endl;

save << wx[i] << endl;

save << wy[i] << endl;

s[i].save(save);

proj[i].save(save);

}

p.save(save);

aoe.save(save);

save.close();

}

**--------------------------------------------------------------------------------------------------------------------------------**

As the **load function**, I add something else as slot 0, which **is a default file, this make sure each time I start the game even within one file fun, It will random a new map seed.**

**--------------------------------------------------------------------------------------------------------------------------------**

bool loadgame(int \_slot, int& level, float\* Game\_time, int\* wx, int\* wy, world\* w, Player& p, Spawn\* s, Projectilemanage\* proj, AOE& aoe) {

ifstream load;

string filename;

filename = "Save/save" + to\_string(\_slot) + ".txt";

if (\_slot == 0) { filename = "Save/startsave.txt"; }

if (fileexist(filename)) {

load.open(filename, ios::in);

load >> level;

if (\_slot == 0) {

w[0].initialrandommappseed();

}

else {

w[level].changemapseed("Save/worldforsave" + to\_string(\_slot) + ".txt");

}

for (int i = 0; i < LEVELNUM; i++) {

load >> Game\_time[i];

load >> wx[i];

load >> wy[i];

s[i].load(load);

proj[i].load(load);

}

p.load(load);

aoe.load(load);

load.close();

return true;

}

return false;

}

**--------------------------------------------------------------------------------------------------------------------------------**

As I save the level, and **all levels data are separated stored** in array, so I can save and load from or to different, **even close and re-open**, it would **load the same random mapseed** as it is part of the save function.

**8.addtional**

I use this value as a base unit of movement, so for it would be influenced by dt value**, the higher the dt is, the higher the unit is**, so the movement looks more smooth since the dt is not constant. However as dt value might various a lot **base on each frame and also different laptop performances**, a fixed value might cause problems such as to fast or too slow. So I use sin value of it to make sure the value is **kept within 0 to 4**.

**--------------------------------------------------------------------------------------------------------------------------------**

float dt = timer.dt(); //get dt value

float u = 2 + 2 \* sin(100 \* dt); //create a unit for moving

**--------------------------------------------------------------------------------------------------------------------------------**

**Template for double linked list**

**--------------------------------------------------------------------------------------------------------------------------------**

template <typename T>

class node {

public:

T data; // data in the node

node\* next = nullptr; // ptr to next node

node\* prev = nullptr; // ptr to previous node

node(T& \_data) : data(\_data) {}

};

template<typename T>

class DBLL {

private:

node<T>\* head = nullptr; // the head of the list

node<T>\* tail = nullptr; // the tail of the list

unsigned int size = 0;

public:

DBLL() {}

~DBLL() {

node<T>\* current = head; // start delete from head

while (current != nullptr) { //if not null, point the head to next node, delete the original head data

node<T>\* next = current->next;

delete current;

current = next;

}

}

//coulde use pointer instead of reference

//add new element at head

void addfront(T& \_data) {

node<T>\* newnode = new node<T>(\_data);

if (head == nullptr) { // empty list

head = tail = newnode;

}

else {

head->prev = newnode; //point from origin head to new

newnode->next = head; //add new before head

head = newnode; // set new as the head

}

size++;

}

// add new element at tail

void addend(T& \_data) {

node<T>\* newnode = new node<T>(\_data);

if (tail == nullptr) {

head = tail = newnode;

}

else {

tail->next = newnode;

newnode->prev = tail;

tail = newnode;

}

size++;

}

//remove an element from the list

void remove(node<T>\* node) {

if (node == nullptr) return;

//last element

if (node == head && node == tail) {

head = tail = nullptr;

}

else if (node == head) {

head = node->next;

if (head != nullptr) head->prev = nullptr;

}

else if (node == tail) {

tail = node->prev;

if (tail != nullptr) tail->next = nullptr;

}

else {

node->prev->next = node->next;

node->next->prev = node->prev;

}

delete node;

size--;

}

void clear() {

node<T>\* current = head;

while (current != nullptr) {

node<T>\* next = current->next;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

size = 0;

}

//find node for the data

node<T>\* find(const T& \_data) {

node<T>\* current = head;

while (current != nullptr) {

if (current->data == \_data) {

return current;

}

current = current->next;

}

return nullptr;

}

//get size of the list

unsigned int getsize() { return size; }

// Get the head of the list

node<T>\* gethead() const { return head; }

// Get the tail of the list

node<T>\* gettail() const { return tail; }

};

**Evaluation**

I tested the game over several run. The double linked list seems to be good enough for the amount of deletion and adding. But it is slow in terms of finding, especially when I go through them doing the AOE detecting, it does have a drop in FPS, but still because I kept a max size for the NPC, the FPS is still quite high. The only big FPS drop I found is that if I increase the canvas size, as for each frame, the amount of pixel needs to be draw increases significantly. Generally the game FPS is maintained at a quite high level.

文本

描述已自动生成图片包含 图示

描述已自动生成

**Limitation and Improvement**

I think the way of detection for collision and detecting the 4 highest health NPCs are too basic. I just went through each object in the double linked list, which has a complexity of O(n). I would consider using other data structures or using some sorting method so it won’t need to go through for every frame but only do that when new objects adding or there is a demage.

Also the Tile class could be get rid justing using the Sprites class.

I should consider using multi files to manage all the classes. And even I tried to make the code easy to be extended, such as new levels or new NPC even Multiple Player. But I only managed to do some of them, still a lot code are defined with fixed value.

Also, to my personal convenience, the encapsulation of classes are very poor, I should pay more attention on that.

The save and load functions works fine, but the file structure is too specific so it requires a lot of change if modification is needed.

I would also consider using smart pointers because a lot of deletion is going on there, to manage the memory more clearly, and less mistakes.

**Conclusion**

I would say I have managed to achieve all the required game functions with something more (not explained here cause they do not contributes to the mark, but I believe they makes it a better game). The performance of the game looks fine to me, all the functions are tested on my best to make sure they work. I have faced a lot of bugs during the development process, but I am glad I have solved them all, hopefully didn’t miss anyone. The report is a general explanation of the coding idea and how I managed to achieve the function. Due to word limitation I couldn’t explain the details of how I progress the code from lots of mistakes. There is a readme file for detailed game instructions(hopefully clear enough). I hope you would enjoy the game!

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Learning Outcome or Evaluative Criteria** | **80+** | **70-79** | **60-69** | **50-59** | **40-49** | **<40** |
|  |  |  |  |  |  |
| **Outstanding** | **Distinction** | **Good Pass** | **Pass** | **Marginal Fail** | **Fail** |
| Master a sound, conceptual understanding of the theory and concepts of programming and fundamental algorithms and data structures. | Demonstrates an exceptional understanding of programming, algorithms, and data structures with highly optimised, efficient, and well-documented code. | Shows a strong understanding of programming concepts and algorithms, with clean and effective code that is mostly optimised | Displays a satisfactory grasp of key concepts, with functional but code which is in parts unoptimised or inconsistent. | Shows basic understanding, with working code that is at times inefficient and demonstrates an overall lack of organisation. | Demonstrates a limited understanding, with poorly structured code that is inefficient and struggles with problem-solving. | Fails to show sufficient understanding or produce functional code, with little to no grasp of programming concepts or algorithms. |
|  |  |  |  |  |  |  |
| Autonomously distinguish the right solution for a given problem from amongst a set of algorithmic and programming tools. | Consistently selects the most efficient and appropriate algorithm or tool for complex problems with clear justification. | Chooses appropriate solutions for most problems, though some decisions could be more optimal or better justified. | Selects suitable solutions for straightforward problems but struggles with more complex scenarios or optimisation. | Makes basic decisions that solve problems but often picks less optimal or inefficient tools. | The correct solution is not always chosen, often selecting inappropriate or inefficient algorithms and tools. | Unable to identify or select appropriate tools for problem-solving, frequently making incorrect or ineffective choices. |
|  |  |  |  |  |  |  |
| Program effectively and independently in a high-level programming language at an intermediate level. | Writes efficient, well-structured, and bug-free code, demonstrating intermediate-level mastery of the language. | Produces effective, mostly error-free code with good structure, showing strong independent programming skills. | Writes functional code independently, though it may contain inefficiencies or structural issues. | Code works but often lacks efficiency or good structure but not ideal problem-solving. | Struggles to write effective code frequently producing inefficient or error-prone solutions. | Unable to independently produce working code at an intermediate level, showing little mastery of the language. |
| Take, straightforward to complex, algorithmic concepts, whether created or based on literature and implement them correctly. | Implements the range of simple and complex algorithms accurately and efficiently, with a deep understanding of underlying concepts. | Correctly implements most algorithms, including complex ones, with minor inefficiencies or occasional gaps in understanding. | Implements straightforward algorithms well but struggles with more complex concepts, resulting in less optimal solutions. | Can implement basic algorithms but has significant difficulty with complex ones, often resulting in incomplete or inefficient solutions. | Struggles to implement even straightforward algorithms correctly, with frequent errors and inefficiencies. | Fails to correctly implement even basic algorithms, showing little understanding of algorithmic concepts. |
|  |  |  |  |  |  |  |