Ice Age Maze Escape Report
CS4303 Video Games Final Game Practical

Author = Joseph Manfredi Cameron

Introduction

Ice Age Maze Escape is a single-player video game of my own creation. An example of a gameplay screen from Ice Age Maze Escape can be seen in Figure 1.

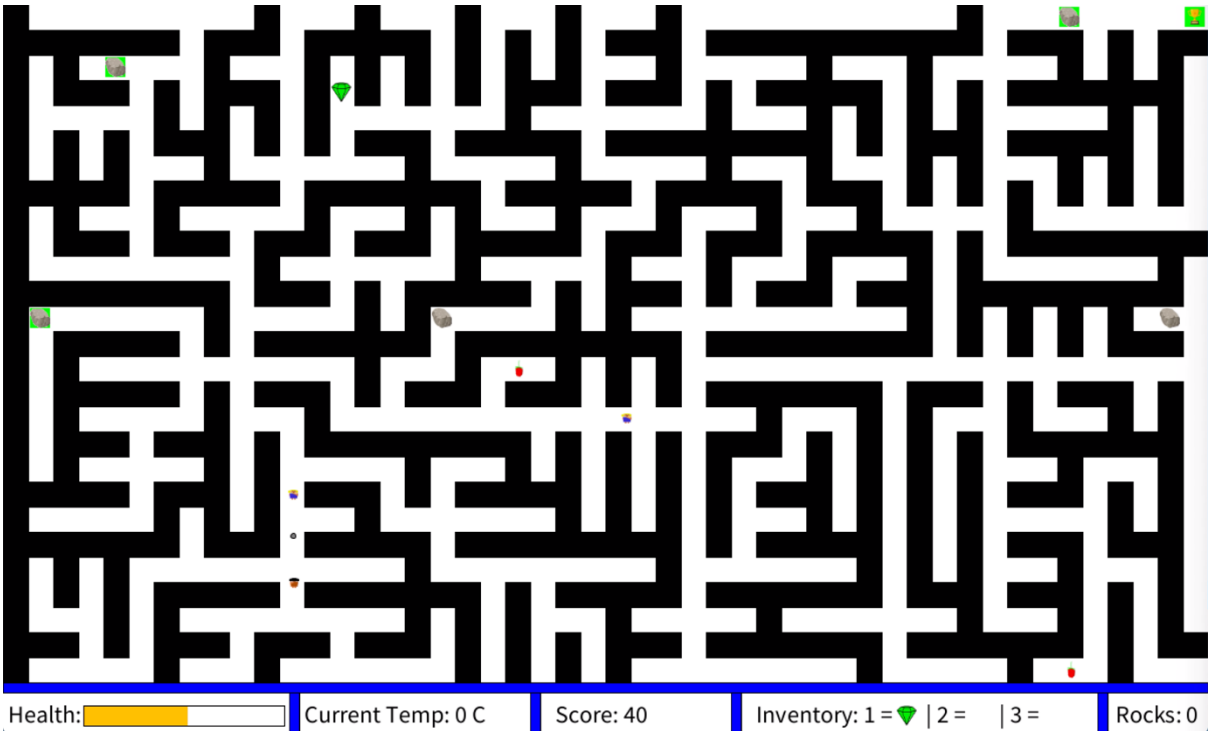

Figure 1: An example gameplay screen from Ice Age Maze Escape

The genre of Ice Age Maze Escape can be best described as a hybrid puzzle, action, and strategy game. The main gameplay of Ice Age Maze Escape consists of a player looking to navigate through a series of mazes for as long as possible before their health runs out. The puzzle element of the game is mainly incorporated via the constant pressure on players to search for paths throughout mazes to get to various objects. Within each maze there are also AI enemies looking to always attack. This is the action part of the game, as players must combat these foes. Strategy plays a massive role in being successful at Ice Age Maze Escape, as players must always manage the items they collect in the maze and plan ahead for future scenarios.

There are noticeable technical capabilities on display in Ice Age Maze Escape. The game contains lots of procedurally generated content, where mazes and placement of objects are procedurally generated, along with various other object and character features such as

power-up type/number per maze or number of enemies per maze. All this procedurally generated content helps to maintain everlasting novelty of gameplay for players, as they are unlikely to ever play the exact same set of mazes with the same objects and characters twice. Furthermore, there is some artificial intelligence present in the game, where there are enemies in the maze that can track the player down and follow them through the maze via A-Star search. Also, there is some interesting physics at work that controls the player's movement in the maze. As the player progresses through the mazes, the temperature gets colder and the ground that the player walks on becomes more and more icy, hence the game starts to simulate varying levels of "slip" in the player's movement due to the ice becoming colder and colder.

Processing [1] was used to develop Ice Age Maze Escape, and the Minim audio library [2] was used to manage and trigger all audio elements of the game. The inclusion of audio plays a significant role in the gameplay and overall gameplay experience for users, as the audio is used to convey important information or reinforce actions within the game.

Game Design

Creating the Game Concept Idea and Game Rules

The idea and concept for the Ice Age Maze Escape video game first came about with my initial curiosity of creating a dynamic maze-based puzzle game. Typically with other games that involve mazes such as Tunnel Runner by CBS Electronics [3] or The Amazing Maze Game by Midway Games [4], the route-finding through the maze itself is the main challenge and there is typically only one route to a single destination that the player is asked to find. Or the map of the maze is typically fixed and unchanging from level to level like we see in Pacman by Namco [5].

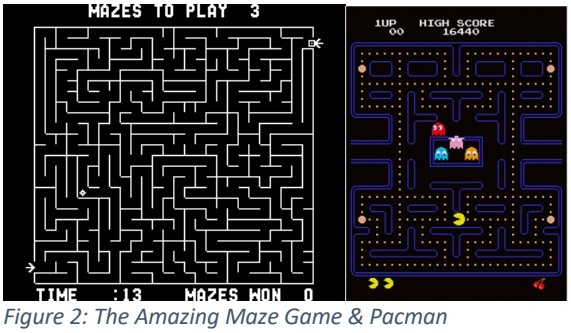

Figure 2: The Amazing Maze Game & Pacman

I wanted to create a game that could challenge players to try and solve many novel/unseen mazes while being forced to look for multiple routes to multiple destinations in each individual maze. I quickly realised that this need for players to find multiple routes in each maze would require the game to offer more options than just finding a single destination from the get-go. At this stage, I came up with the idea to include an ever-decreasing health bar for the player and then include health boosting objects in the maze that the player would have to move to restore their health. Suddenly, this concept makes a maze game

more fun and interesting because now the player is required to think about more than just a single destination, they must think about multiple destinations and judge their health at the same time. Plus, there is the time pressure present that keeps players engaged with the game.

I decided to run with this idea of asking players to consider and judge multiple routes and variables in mazes as it makes gameplay far more exciting and dynamic. Hence, I decided to introduce special “protected” objects into the game idea that players cannot initially interact with, instead they would be required to collect another separate item to then pick up the “protected” object. I thought it would be interesting to make the goal destination of the maze sometimes “protected”, and so I came up with the concept of players having to collect trophies to progress from one maze to another, and sometimes the goal trophy would be “protected” which would force the player to first consider the route to get the special object that allows them to collect the protected trophy. Meanwhile, the time pressure from ever-decreasing health and having to find health boosters is constantly prevalent as well.

At this stage, I thought it would also be meaningful to cause damage to the player if they touch a wall of the maze. My reasoning for this is that when players are put under pressure to quickly scan multiple points of a maze, it would be quite easy for them to not always focus on the player character but instead look elsewhere. Hence, the fact that there would be damage upon touching a wall adds another layer of complexity to the game. Then, the whole concept of setting the game in the Ice Age came to me, as I thought it would be cool to implement ice physics into the player’s movements which would get progressively more “slippy” as the game progresses and cause more jeopardy with regard to the player hitting walls. At this point the name Ice Age Maze Escape was born.

I then updated the health decreasing concept to include the fact that the rate of change would increase for every level/maze that the player faces. The idea is that if the temperature were to get colder and colder, then the rate of the player’s health decreasing would increase. This element along with the ice physics increasing for every level are the main factors for increasing difficulty of the game as the player progresses.

Subsequently, I realised that the next element that would make for a complete game would be the inclusion of enemy characters that exhibit some form of artificial intelligence, similar to the enemies in Pacman [5]. So, I decided to the make the player character a caveman of a certain tribe (in tune with the Ice Age theme), and the enemy characters would be cavemen of another tribe looking to attack the player as they try and progress through the mazes. Along with this addition of enemies came the idea of adding weapons and power-ups to the mazes. The weapons would be placed throughout the maze and could be used to kill enemies who would otherwise inflict damage on the player’s health. This is another strategic element players must now consider in the game. I decided to include two power-ups, one which kills all enemies in any given maze instantly, and one that shows the player the correct route to the goal trophy. These power-ups seemed the most natural/useful to include for this game concept.

Finally, I decided to add a scoring system to the game where points would be awarded to the player for both collecting trophies and killing enemies. The reason for this is that the points would indicate how much exploration of the mazes a player has done. For example, a player who has focused on immediately collecting trophies to progress through mazes would score less points than an exploratory player who has killed enemies and collected items throughout the same number of mazes.

Main Menu & Game Controls Screens

When a player first starts Ice Age Maze Escape, they are presented with the main menu screen shown in Figure 3.

Figure 3: Ice Age Maze Escape Main Menu

This main menu gives the players some simple instructions on how to either start a new game or access the game controls screen. I wanted to keep the game instructions simple to try and highlight the gameplay above everything else. My goal is to get the player playing as fast as possible.

Also, this main menu screen sets the tone for the whole game, with the blue and white colour combination that is consistent across all game screens, and the setting of the game in the Ice Age. Furthermore, there is a looping audio track playing music that hopefully further pushes the player to be in the mood to play the game.

If a player pushes the '1' key, they are taken to the game controls screen that is shown in Figure 4.

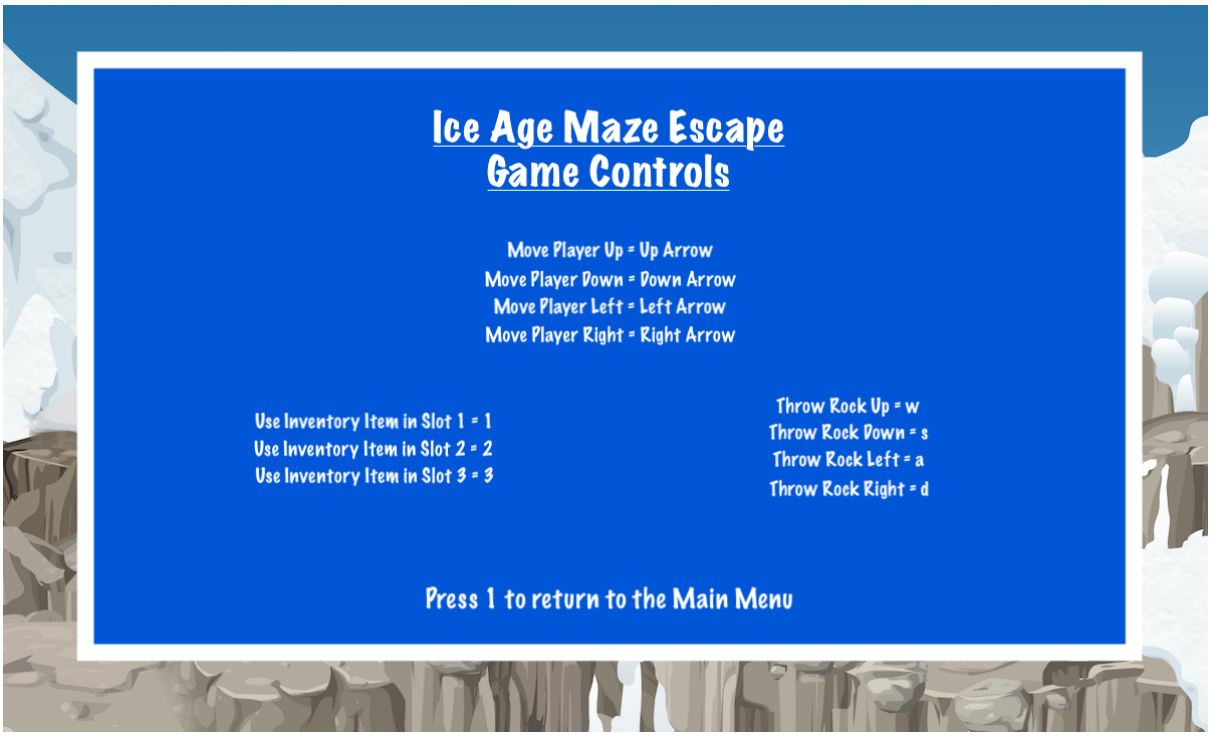

Figure 4: Ice Age Maze Escape Controls Screen

It is immediately clear that the aesthetic that was established with the main menu is continued through to the controls screen (and in fact all other screens). This gives the game a consistent and integral identity.

The controls scheme is designed to designate each hand of the player to a particular task. The right hand oversees movement, and the left hand oversees throwing weapons/rocks and using inventory items. In future work, I would look to add a mechanism that allows players of different handedness (right or left handed) to customise these controls.

Having the controls displayed in-game is also very handy for people who have not seen the player's guide but want to start playing in the application!

Gameplay Screen

With the idea and design schemes of Ice Age Maze Escape now flourishing, I created a quick sketch, shown in Figure 5 below, of what I initially wanted the gameplay screen to look like.

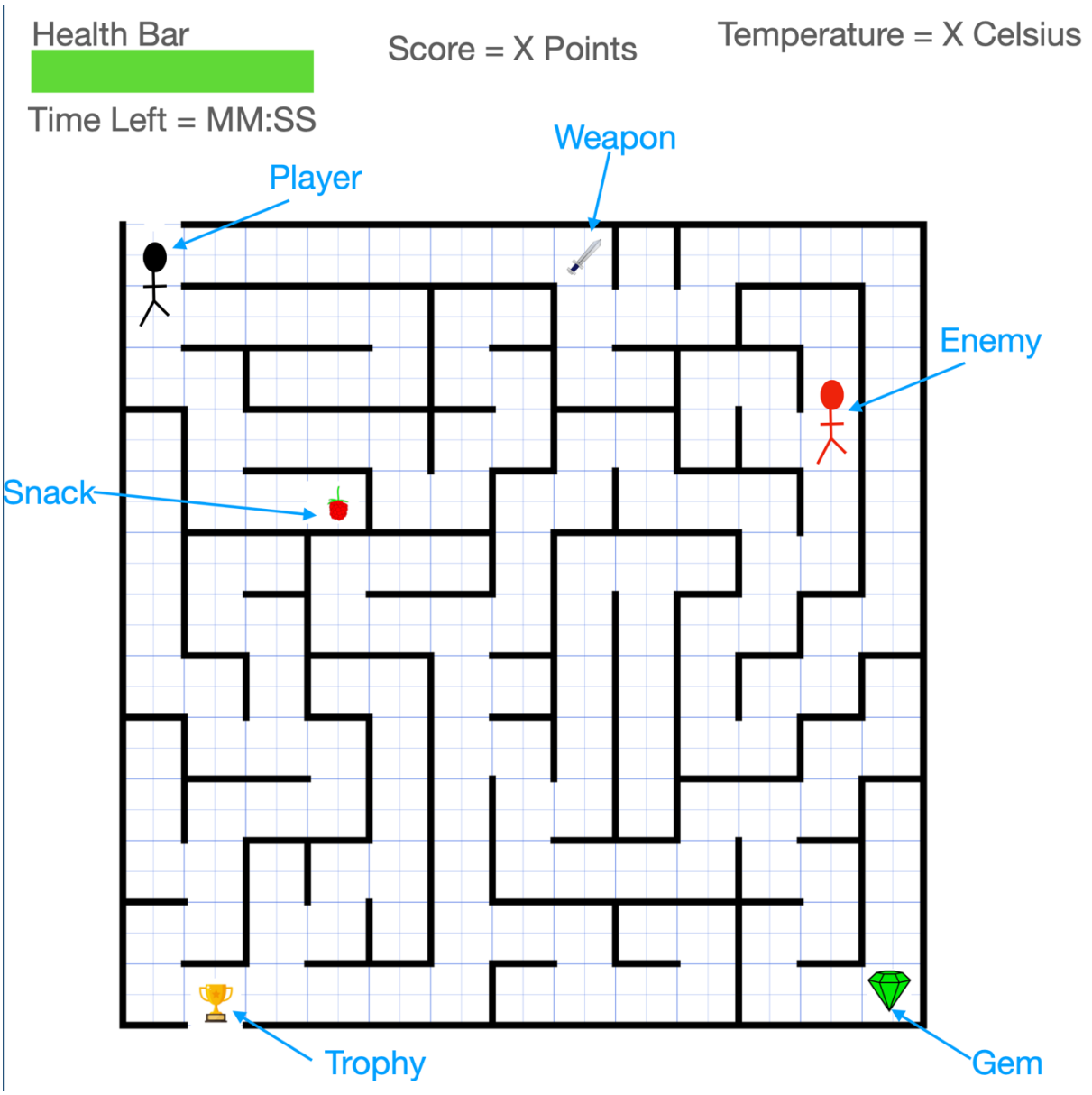

Figure 5: An early sketch for the gameplay screen design of Ice Age Maze Escape

The sketch shows my initial idea for a game screen with a top-down view on the entire maze and the various items located within it. The health-boosting items that were mentioned earlier are now known as snacks shown by the raspberry. The special object that grants players with the ability to pick up protected objects are now gems shown by the green gem.

Following this sketch, I became curious about researching different views for the Ice Age Maze Escape game. This initial idea is a top-down view where the player can immediately see everything in a maze, but I thought it would be useful to explore a different view where the player is always fixed at the centre of the screen and the view is zoomed in on them, hence the player cannot see the whole maze but only a portion of it. I created a sketch of this that can be seen in Figure 6, and then thought through the processes that this new view would allow.

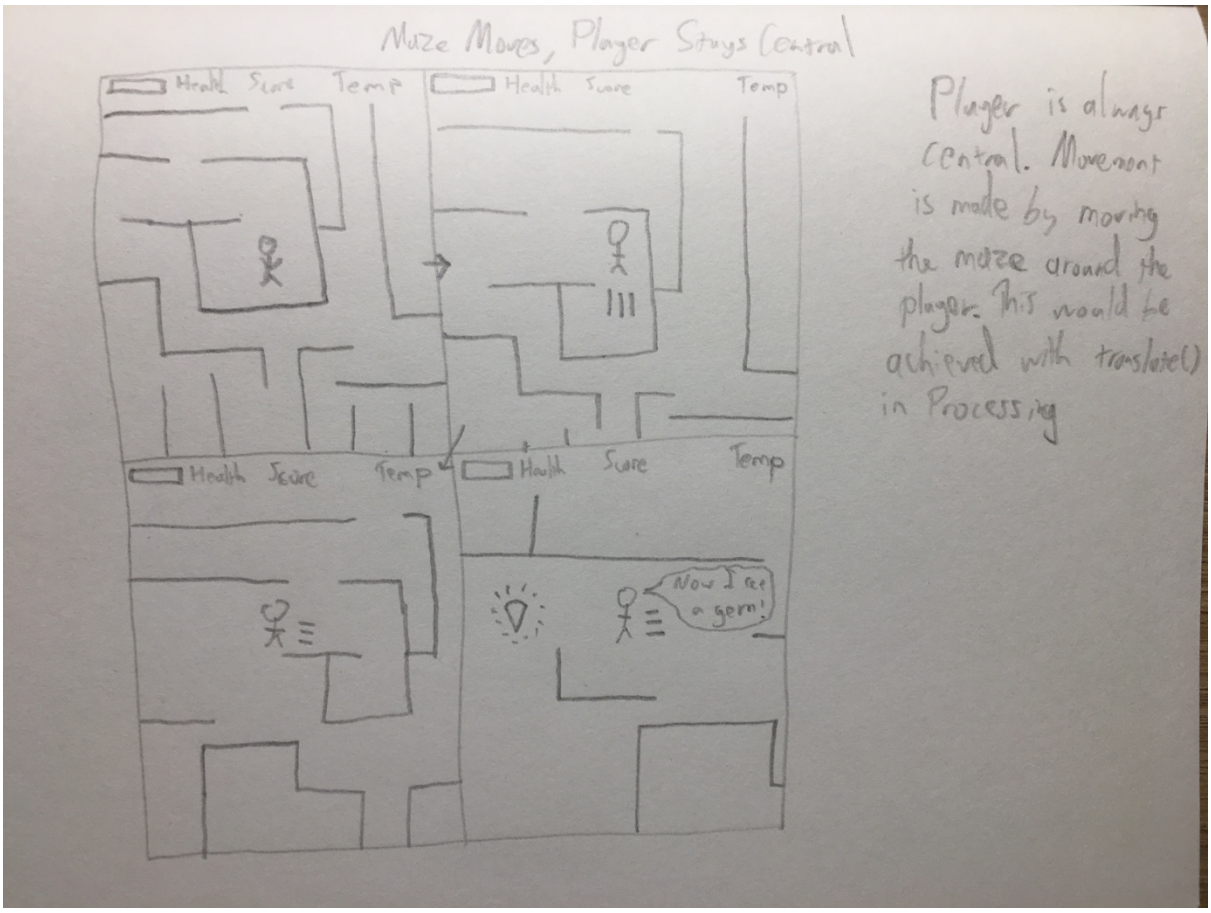

Figure 6: Alternate view of the gameplay screen for Ice Age Maze Escape

As the sketch in Figure 6 shows, this zoomed-in view of the player in the maze would require entirely different strategy and movement mechanics. With this game screen view, instead of the player moving around the screen, the player would instead stay fixed in the centre of the screen and the maze would be moving around the player in order to portray movement throughout the maze. This action is achieved in Processing by using the translate() function [6]. This zoomed-in view would also change the strategy of the game, as the player can no longer see all items in the maze and plan ahead like they can with the top-down view. The sketch shows a situation where the player aimlessly wanders and then stumbles across a gem. This view could provide cool gameplay as it would place the player in the maze in similar fashion to how it would feel in real life.

At this stage I was at a crossroads for deciding which design would result in a better game as both views could potentially result in very different games respectively, so I made low-fidelity prototypes of both ideas in Processing. After playing with both ideas, I concluded that the top-down view of the overall maze was by far the better option. Although the concept of zooming in on the player in the maze and having an exploratory feel for gameplay is an intriguing prospect, it simply turned out to be too frustrating to play with in practice, as that view did not align well with my initial game ideas. For example, gameplay can begin to feel extremely frustrating as soon as there any time pressure involved when playing with the zoomed in view. Additionally, if the player must also consider retrieving vital items that they cannot see along with the time pressure, that is a recipe for an extremely annoying game. My initial idea of involving strategic gameplay where players can

plan multiple routes is a fundamental pillar holding up the whole concept of Ice Age Maze Escape, and the original top-down view I sketched successfully delivers that concept to the player far better than the other zoomed-in view. Hence, I decided to scrap the idea of the zoomed-in view and instead move forward with my original top-down view concept.

Next, I iterated my design of what the gameplay screen should look like in order to better improve it for the needs of the player during this game. Eventually, I settled on the gameplay screen design that can be seen in Figure 7 below for the final version of Ice Age Maze Escape.

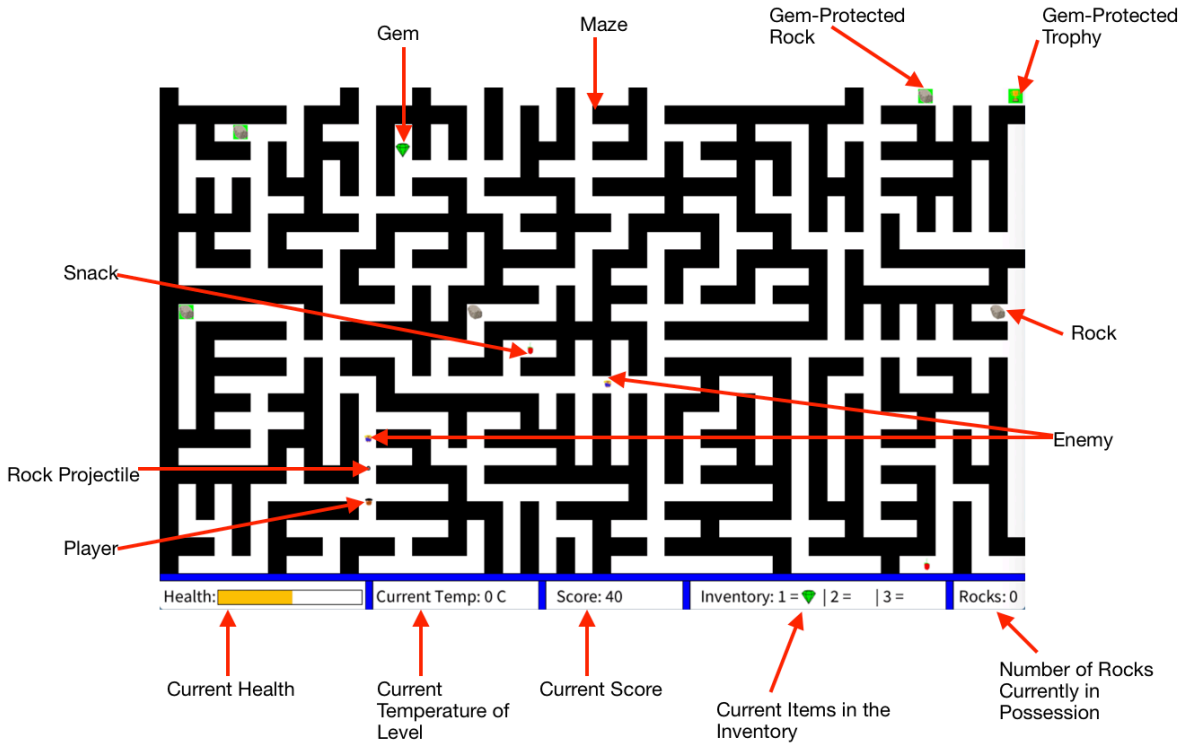

Figure 7: Improved Gameplay Screen Design for Ice Age Maze Escape

This final version of the game screen design is an improvement on the original sketch in Figure 5 because this new design includes many more crucial statistics that players need to know during the game, such as what items they currently have in their inventory and how many rocks are currently in their possession.

Game Summary Screen

Between every level of gameplay in Ice Age Maze Escape, I felt it was very important to include a summary screen that shows the player's progress through the game so far along with any other useful current player statistics. An example of the game summary screen can be seen below in Figure 8.

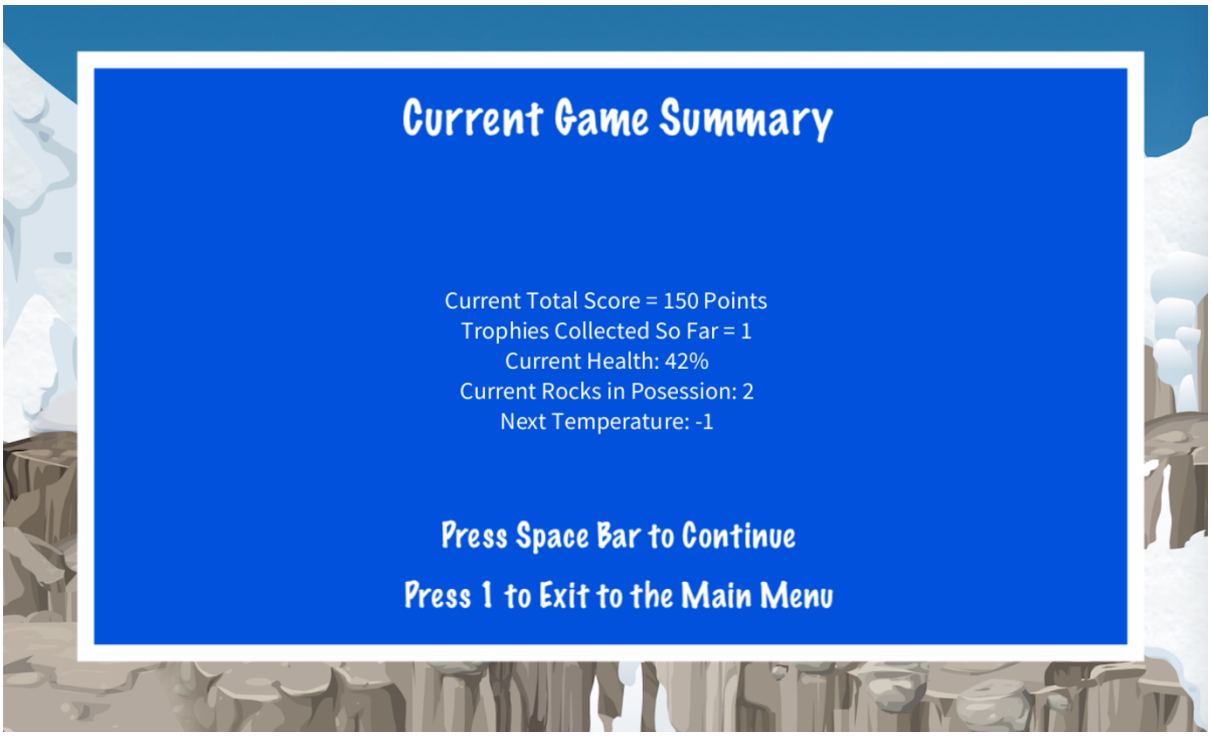

Figure 8: Ice Age Maze Escape Game Summary Screen

When a player collects a trophy, it didn't feel right to immediately throw them back into another new maze. This summary screen also acts as a vital breather for the player to relax before continuing. Successfully completing a maze in Ice Age Maze Escape can require lots of mental focus with all the tasks a player may need to give attention to, so this breather is an important component of keeping gameplay fun instead of overly intense.

Game Over Screen

After a player loses all their health, the game ends and then they are shown the game over screen. An example of the game over screen can be seen below in Figure 9.

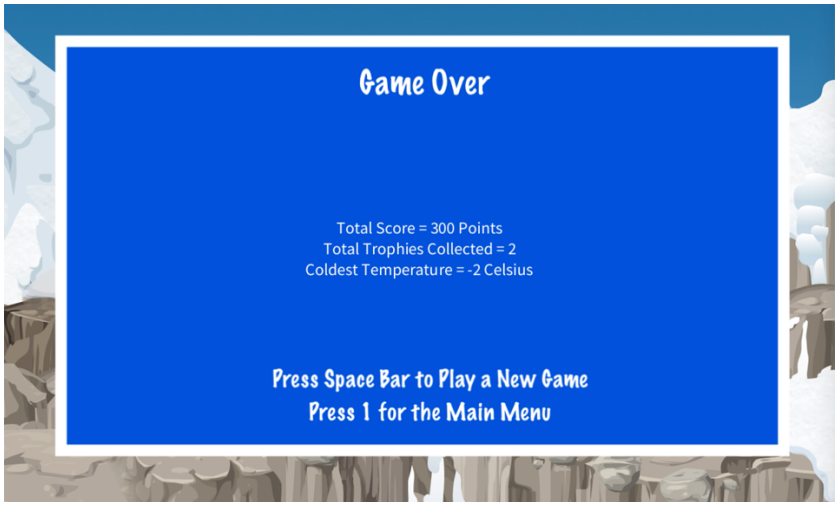

Figure 9: Ice Age Maze Escape Game Over Screen

When this screen is shown, a sad sounding audio sample is played to further reinforce the sad connotations with the scenario of game over. The player's final score, trophies collected, and coldest level are shown along with instructions on how to start a new game or go back to the main menu.

In Ice Age Maze Escape, there is a special scenario which might lead to an unexpected game over if the player is not paying close attention. Since trophies can be gem-protected, it is possible for a player to be stranded in a maze with no way of progressing because they have incorrectly used their gems for picking up other items instead of prioritising the trophy. If this scenario happens, then the game over screen in Figure 10 below is shown to the player instead.

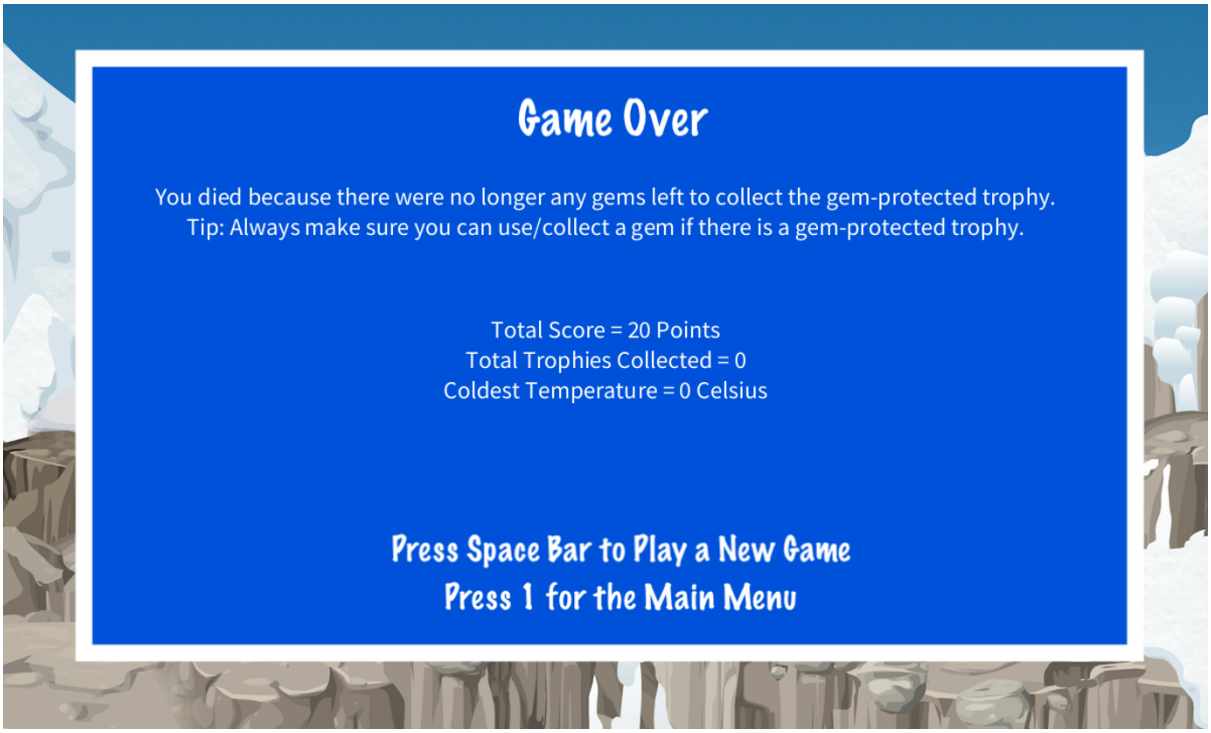

Figure 10: Ice Age Maze Escape Game Over Screen with Additional Tip Message

The addition of this message in the game over screen helps the player to become aware of this strategic mistake and avoids any potential confusion.

Audio

Ice Age Maze Escape has plenty of audio samples playing throughout that enhance gameplay and overall user experience. All audio is strategically used to either reinforce game actions, or simply to set a mood for the player. The main menu music was composed by me in Logic Pro X [7] and provides the perfect tense atmosphere I want players to feel before playing the game. A notable audio sample I used in the game when an enemy character runs into the player is the famous and recognisable Wilhelm scream. This is a famous open domain stock sound used commonly in Hollywood cinema. All other audio samples were obtained from the Apple Loops sample library found in the Logic Pro X application [8].

Game Implementation

Before diving into the implementation details of Ice Age Maze Escape, here is an overview UML class diagram of all the Java Code for the entire Ice Age Maze Escape Processing project.

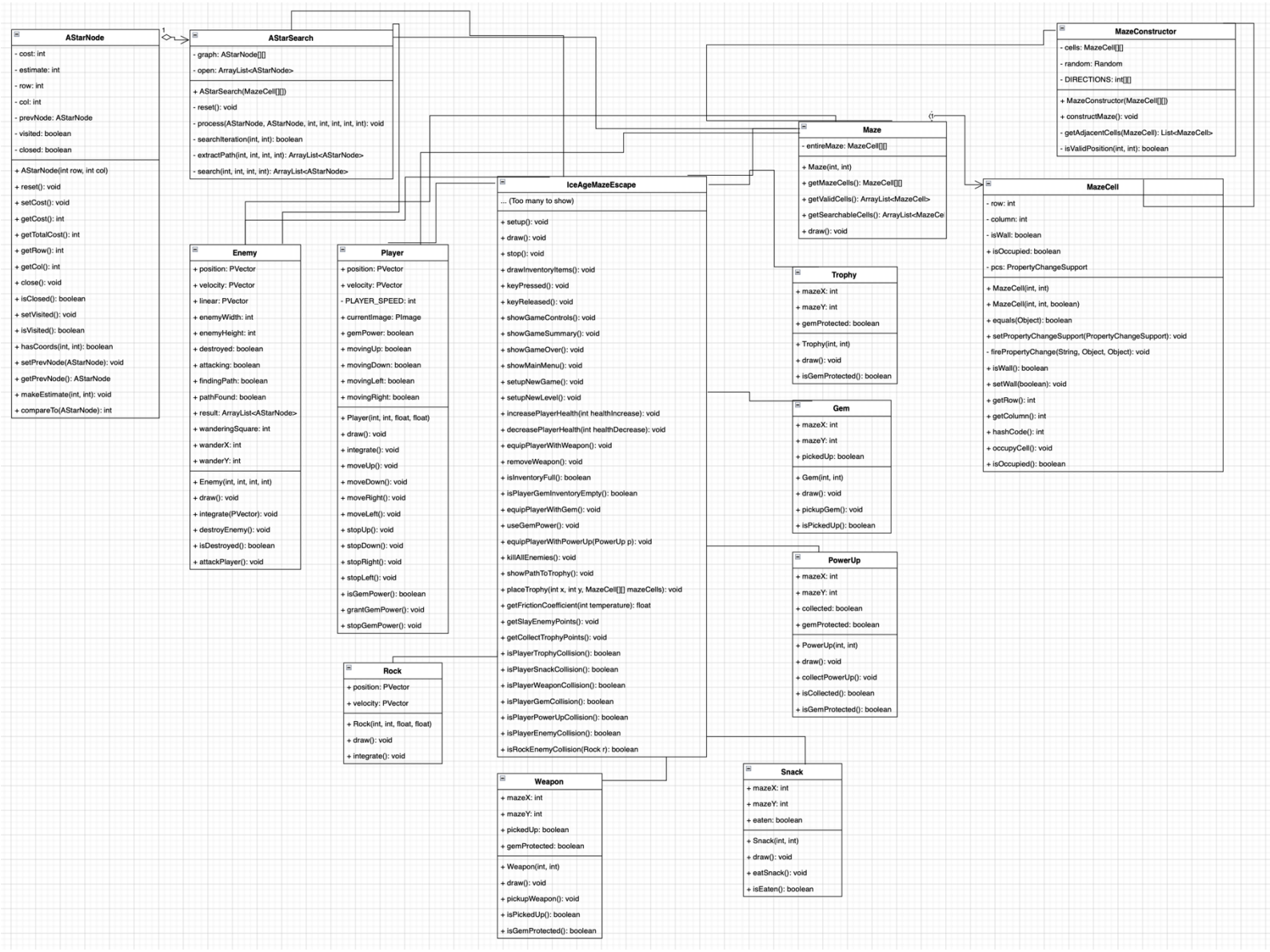

Figure 11: Ice Age Maze Escape UML Class Diagram

Game State

The game state of Ice Age Maze Escape is managed by five main Boolean variables on the main game file in the Processing sketch (IceAgeMazeEscape Class). These Booleans control which of the five different screens (main menu, controls screen, gameplay screen, game summary screen, or game over screen) are shown to the player. Only one of these Booleans is true at any given time because only one screen can be shown at any given time.

Gameplay Setup

Every aspect of a new maze is set up before the start of a level. Mazes, objects, characters, and their positions are all procedurally generated at this prior stage. Find out more in the "Procedurally Generated Content" section below. Certain statistics carry over from maze level to maze level, such as the player's health, the inventory, the number of rocks in the player's possession etc. When a new game is started, all statistics and settings, including those previously mentioned, are reset to their default settings.

Procedurally Generated Content

A significant portion of the Ice Age Maze Escape game is procedurally generated. This is a very important feature of Ice Age Maze Escape as it is what ensures that players are continuously playing in novel mazes, with a random assortment of objects and characters positioned randomly on those mazes. This procedural generation is one of the best strategies for keeping players constantly engaged and excited to play the game.

All mazes within the game are procedurally generated with a method based on Prim's algorithm [9]. My code (located in the MazeConstructor class) to achieve this can be seen in Figure 12 below. All code for generating mazes can be found in the Maze, MazeCell, and MazeConstructor classes.

```
// Construct the maze using a method inspired by Prim's Algorithm (https://en.wikipedia.org/wiki/Maze_generation_algorithm)
void constructMaze() {

  //Start with a grid full of maze cells that are walls
  for(int i = 0; i < cells.length; i++){
    for(int j = 0; j < cells[0].length ; j++){
      cells[i][j].setWall(true);
    }
  }

  //Pick a random cell
  int x = random.nextInt(cells.length);
  int y = random.nextInt(cells[0].length);

  // Get starting point for the player to be positioned in
  mazeStartX = x;
  mazeStartY = y;

  // Set this initial cell to the first valid cell
  cells[x][y].setWall(false);

 // Get adjacent cells
 Set<MazeCell> adjacentCells = new HashSet<>(getAdjacentCells(cells[x][y], true));
 // Keep getting and looking at cells until they have all been visited
 while (!adjacentCells.isEmpty()){
   // Select a random cell from the adjacent cells
   MazeCell randomAdjacentCell = adjacentCells.stream().skip(random.nextInt(adjacentCells.size())).findFirst().orElse(null);
   // Retrieve the adjacent non-wall cells of the random adjacent cell above
   List<MazeCell> adjacentNonWallCells = getAdjacentCells(randomAdjacentCell, false);
   // Make a route to the adjacent non-wall cells
   if(!adjacentNonWallCells.isEmpty()) {
     MazeCell routeCell = adjacentNonWallCells.get(random.nextInt(adjacentNonWallCells.size()));
     int connectingRow = (routeCell.getRow() + randomAdjacentCell.getRow())/2;
     int connectingCol = (routeCell.getColumn() + randomAdjacentCell.getColumn())/2;
     randomAdjacentCell.setWall(false);
     cells[connectingRow][connectingCol].setWall(false);
     routeCell.setWall(false);
   }
   // Get more adjacent cells for other cells around the maze to continue the search
   adjacentCells.addAll(getAdjacentCells(randomAdjacentCell, true));
   // Finally, remove this current cell as it has been visited
   adjacentCells.remove(randomAdjacentCell);
 }
}
```

```
// Retrieves all valid surrounding cells
private List<MazeCell> getAdjacentCells(MazeCell cell, boolean isWall) {
  List<MazeCell> adjacentCellList = new ArrayList<>();
  for (int[] direction : DIRECTIONS) {
    int newRow = cell.getRow() + direction[0];
    int newCol = cell.getColumn() + direction[1];
    if (isValidPosition(newRow, newCol) && cells[newRow][newCol].isWall() == isWall) {
      adjacentCellList.add(cells[newRow][newCol]);
    }
  }
  return adjacentCellList;
}

// Returns true if the row and column values equal a valid position
private boolean isValidPosition(int row, int col) {
  return ((row >= 0) && (row < cells.length) && (col >= 0) && (col < cells[0].length));
}
```

Figure 12: My code to procedurally construct mazes with Prim's Algorithm

Essentially with Prim's algorithm, I first make all the maze cells walls, pick a random starting point (which I also keep as a starting point for the player), make the starting cell a non-wall cell and then pick another random adjacent cell. I keep doing this recursively for every adjacent cell until every cell has been visited. At the end, there is a maze. With this procedural generation, I can generate completely novel mazes for every level of gameplay.

There are also many other features that are procedurally generated in Ice Age Maze Escape. These include the quantity of snacks, rock weapons, enemies, gems, and power ups. Furthermore, the maze placements of these items are also randomly generated. The type of power-up is randomly determined when a power-up object is instantiated. Furthermore, all items that can be gem-protected (trophies, rock weapons, power-ups), are also randomly determined to be either regular or gem-protected when they are instantiated. An example of this implemented can be seen in my Trophy class constructor method shown below in Figure 13.

```
Trophy(int x, int y) {
  mazeX = x;
  mazeY = y;
  int protectionChance = (int) random(1, 4); // 1 in 3 chance of protected trophy
  if (protectionChance == 3) {
    gemProtected = true;
  } else {
    gemProtected = false;
  }
}
```

Figure 13: The Trophy Class Constructor Method

An interesting implementation feature to look at is my method of placing trophies. To place a trophy, I strategically always place it as far away from the player's starting point as possible to always ensure a reasonably challenging maze level for the player. My function to achieve this can be seen below in Figure 14.

```
// Given a xy coordinate, this finds the furthest xy coordinate in the maze to place trophy
void placeTrophy(int x, int y, MazeCell[][] mazeCells) {
  float maxDistance = 0;
  ArrayList<MazeCell> validCells = new ArrayList<MazeCell>();
  //Find all maze cells that are not walls or occupied
  for(int i = 0; i < mazeCells.length; i++) {
    for(int j = 0; j < mazeCells[0].length; j++) {
      if ((!mazeCells[i][j].isWall()) && (!mazeCells[i][j].isOccupied())) {
        validCells.add(mazeCells[i][j]);
      }
    }
  }
  // Now, find the furthest cell from x and y
  for (int a = 0; a < validCells.size(); a++) {
    float distance = dist(x, y, validCells.get(a).getRow(), validCells.get(a).getColumn());
    if (distance > maxDistance) {
      maxDistance = distance;
      trophyX = validCells.get(a).getRow();
      trophyY = validCells.get(a).getColumn();
    }
  }
}
```

Figure 14: The placeTrophy function from the main game file

Game AI

The enemy characters present in the mazes have artificial intelligence implemented. They find routes around mazes via A* Search, and the code to implement this was taken from StudRes [10]. In Ice Age Maze Escape, this code can be found in the AStarNode and AStarSearch classes. Of course, I tweaked the StudRes code to fit into Ice Age Maze Escape, the code comments fully illustrate the changes I made. I must also mention that the overall ideology of how the artificial intelligence was implemented was very much based on the work I completed in practical 2 of this course, where I was developing Robotron4303. Hence, much of the AI code structure is similar to that implementation as well.

The enemy characters are always initially in ‘exploratory’ mode, so their A* path finding leads them towards randomly chosen cells of the maze, until they reach either the same row or column as the target cell, where subsequently a new path towards a new maze cell is found. This behaviour perfectly mimics the expected ‘exploratory’ behaviour. When the enemies detect a player within 125 pixels of their location, they then switch to attack mode where their A* path finding switches to the player’s location continuously. The code that achieves this is shown in Figure 15.

```
if (attacking) {
  result = pathFinder.search((int) (position.x/25), (int) (position.y/25), (int) (targetPos.x/25), (int) (targetPos.y/25));
} else {
  // Wander
  // Only need to wander to the same row or column, this helps preserve 'patrolling' behaviour
  if ((((int) (position.x/25)) != wanderX) && (((int) (position.y/25)) != wanderY)) {
    result = pathFinder.search((int) (position.x/25), (int) (position.y/25), wanderX, wanderY);
  } else {
    wanderingSquare = (int) random(0, maze.getSearchableCells().size());
    wanderX = ((int) maze.getSearchableCells().get(wanderingSquare).getRow());
    wanderY = ((int) maze.getSearchableCells().get(wanderingSquare).getColumn());
    result = pathFinder.search((int) (position.x/25), (int) (position.y/25), wanderX, wanderY);
  }
}
```

Figure 15: Enemy AI Code

Player Movement Physics

There is an interesting physics implementation in Ice Age Maze Escape where the ground is simulated to act more and more like ice as the temperature gets colder, hence the player starts to slide more and more while moving as a result. The code to achieve this is shown below in Figure 16.

```
void integrate() {

  if (movingUp) {
    velocity.y = -PLAYER_SPEED;
    currentImage = playerImages[1];
  } else if (movingDown) {
    velocity.y = PLAYER_SPEED;
    currentImage = playerImages[0];
  } else {
    if ((velocity.y >= 0.01) || (velocity.y <= 0.01)) {
      velocity.y = (velocity.y/(2/getFrictionCoefficient(temperature)));
    } else {
      velocity.y = 0;
    }
  }

  if (movingLeft) {
    velocity.x = -PLAYER_SPEED;
    currentImage = playerImages[2];
  } else if (movingRight) {
    velocity.x = PLAYER_SPEED;
    currentImage = playerImages[3];
  } else {
    if ((velocity.x >= 0.01) || (velocity.x <= 0.01)) {
      velocity.x = (velocity.x/(2/getFrictionCoefficient(temperature)));
    } else {
      velocity.x = 0;
    }
  }
```

Figure 16: Code from Player Class that simulates the physics of sliding on ice

Whenever the player is not moving, their velocity is continuously being reduced by a certain rate. To simulate ice, the rate of the velocity slowing down is simply reduced so it takes longer for the player's velocity to reduce. The getFrictionCoefficient function returns a value between 1 and 2 that trends closer to 2 as the temperature decreases to better simulate more and more slide on the ground due to ice.

Game Evaluation

One of my strongest evaluation techniques was simply to play Ice Age Maze Escape many, many times. As a result, I was able to thoroughly test and evaluate the game.

Initially, testing involved making sure that the procedurally generated mazes were correct, meaning that there always valid mazes where players can navigate to trophies and all the other items that are available. After playing hundreds of times after full implementation, I always found the procedural generation for mazes and items to be correct. However, something I had overlooked was that sometimes players can be stranded if they can't use gem power to collect a gem-protected trophy. Hence, I went back to my implementation and included the check for this scenario that also adds the tip message in the game over screen that was mentioned earlier in the "Game Design" section of this report.

Most of my time on evaluation for Ice Age Maze Escape was dedicated towards perfecting the parameters of multiple components along with getting the scoring system balanced. This was crucial to obtain enjoyable and fun gameplay for players. The main parameters that needed tweaking for enjoyable gameplay were the friction coefficients to start introducing the right amount of slide to simulate ice in the player's movement physics, the value of health boost that snacks provide, the number of enemies present in mazes, and the rate of player health decreasing for various temperatures. Eventually after much tweaking and changing the values for these parameters in the code and then playing the game again, I settled on a good balanced game that offers a stern challenge to players, but in a fun way that is not too difficult.

Game Context

The most similar game to Ice Age Maze Escape that is well known by most of the general population would be Pacman [5], mainly for the fact that Pacman is also set in a maze. Another similarity between Pacman and this game is the artificial intelligence behaviour of the enemy characters, who chase the player around the maze once they have "detected" the player.

An additional similarity that Ice Age Maze Escape has to other maze games is the element of time pressure to complete the mazes. In Ice Age Maze Escape, the time pressure comes in the form of the player's health decreasing. In similar games such as Tunnel Runner [3] and The Amazing Maze Game [4], the time pressure is more explicit in the form of a clock.

Along with the similarities mentioned above, there are also lots of unique elements to Ice Age Maze Escape where it differs from all the other games. As far as I am aware, the requirement of players to map multiple routes to multiple destinations in mazes to maintain health or collect items is a unique feature that illustrates much more dynamic gameplay where players can get very focused compared to an otherwise more static maze game. When evaluating Ice Age Maze Escape, I found myself getting so focused with the game that time would fly by quicker than usual. I also found that when high pressure situations occur

in Ice Age Maze Escape due to multiple things happening at once, I would start getting the feeling of a little bit of adrenaline. This shows a level of immersion going on with the gameplay. Another very unique feature with Ice Age Maze Escape that I haven't seen in other maze games is the implementation of health damage when a player hits a maze wall, along with the inclusion of the ice physics on the ground that encourages players to "slip" into the maze's walls. This feature adds an element of precision and accuracy that a player would normally find on a racing game into this genre of puzzle and action maze game.

Conclusion

Overall, it has been incredibly rewarding to create my own game during this project! The concept of Ice Age Maze Escape turned out to be very novel and immersive, with its own identity, feel, and place among puzzle action games. The implementation also did an exceptional job of correctly conveying the game concept and made for a fun gameplay experience with players.

As the evaluation showed, the parameters within which Ice Age Maze Escape is played will always require further tweaking to better achieve a balanced game for different skill levels and abilities. A key area of future work in this game would be exactly that. I would investigate how different players cope with different parameters on the mazes, objects, enemies, ice physics etc. and then start attempting to build a robust difficulty system that can allow the difficulty of the game to better suit lots of different players around the world.

Works Cited

[1] "Processing," Processing Foundation, [Online]. Available: https://processing.org. [Accessed 08 May 2022].

[2] "Minim Audio Library for Processing," [Online]. Available: https://code.compartmental.net/minim/. [Accessed 08 May 2022].

[3] *Tunnel Runner.* [Art]. CBS Electronics, 1983.

[4] *The Amazing Maze Game.* [Art]. Midway Games, 1976.

[5] *Pacman.* [Art]. Namco, 1980.

[6] "Processing Reference - Translate Function," Processing Foundation, [Online]. Available: https://processing.org/reference/translate_.html. [Accessed 08 May 2022].

[7] "Logic Pro," Apple Inc., [Online]. Available: https://www.apple.com/uk/logic-pro/. [Accessed 08 May 2022].

[8] "Apple Loops in Logic Pro," Apple Inc., [Online]. Available: https://support.apple.com/en-gb/guide/logicpro/lgcp734a05f6/mac. [Accessed 08 May 2022].

[9] "Prim's Algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Prim%27s_algorithm. [Accessed 08 May 2022].

[10] "A Star Search Code from Studres," [Online]. Available: https://studres.cs.st-andrews.ac.uk/CS4303/Lectures/W5/CS4303_2022_Wk5_L3_AI4/Astar_exercise/AStarSoln/. [Accessed 08 May 2022].