CS4303 Practical 2 – AI: Robotron

200022530 – Due: 15/03/2024

Overview

A variant of the classic video game Robotron 2084, an arcade-style twin-stick shooter, programmed in processing. The player must fight against the swarms of robots in each wave and save as many humans as possible, with each wave getting progressively harder. The game has the following features, including all those outlined in the spec, as well as some of my own additions.

- The player can move and shoot bullets at robots, with collision. Movement and shooting are both controlled by keys – **wasd for movement and the arrow keys for shooting**.
    - When the player is first spawned onto the stage in the spawn quadrant, they are given brief invincibility. As well as receiving **brief invincibility when the player is hit by a robot**.
    - Robots and humans cannot spawn in the spawn quadrant.
- There are **procedurally generated** player area of rooms and corridors. Rooms are generated by splitting the screen area into quadrants randomly and connecting them with corridors.
- There are **4 types of robots**[1] for the player to attack:
    - *Red robots* attack the player.
    - *Orange robots* are hunters, they hunt to kill humans. When there are no more humans, they will attack the player.
    - *Purple robots* are infectious, they will aim to infect humans to turn them into *infected robots*. When there are no more humans, they will attack the player.
    - *Infected robots* are visually identical to red robots and will attack the player, however they have a higher movement speed.
- There are **3 types of humans** for the player to save, children, teenagers, and adults. These humans vary in size, such that children are the smallest and adults are the largest. The player will receive **more points for saving younger humans**.
- Humans and robots will explore the stage using the **kinematic algorithms from the lectures**, unless they are pursuing something (or in the case of humans, pursuing or fleeing). Humans and robots will pursue the player when they see them, to kill them or be saved by them. Humans will run away from purple and orange robots to the edges of rooms, waiting to be saved.
    - **All entities employ A* search** in the planning of its movements.
- There are **2 power ups** that will spawn randomly on the stage and appear as small squares:
    - Life Up will add one life to the player's current lives.
    - Invincibility will make the player invincible for a short period. When the player is invincible, it will appear grey.
- **Scores are added for robots that the player destroys and humans the player saves.**
- A HUD (Heads-Up Display) shows relevant information to the player such as score and wave number. Additionally, between waves the HUD will show an indicator that a new wave has started.
- **Hazards/Obstacles** are randomly generated in rooms, they are destructible and on contact will cause the player to lose a life.
- Players can interact with a title and lose screen.
- Included Player Guide made using Canva[2].

[1] Colours were chosen to work best with my colour blindness – protanopia. More distinct colours may exist for people with full colour vision.
[2] (Canva, Accessed: 15/03/2024)

Design

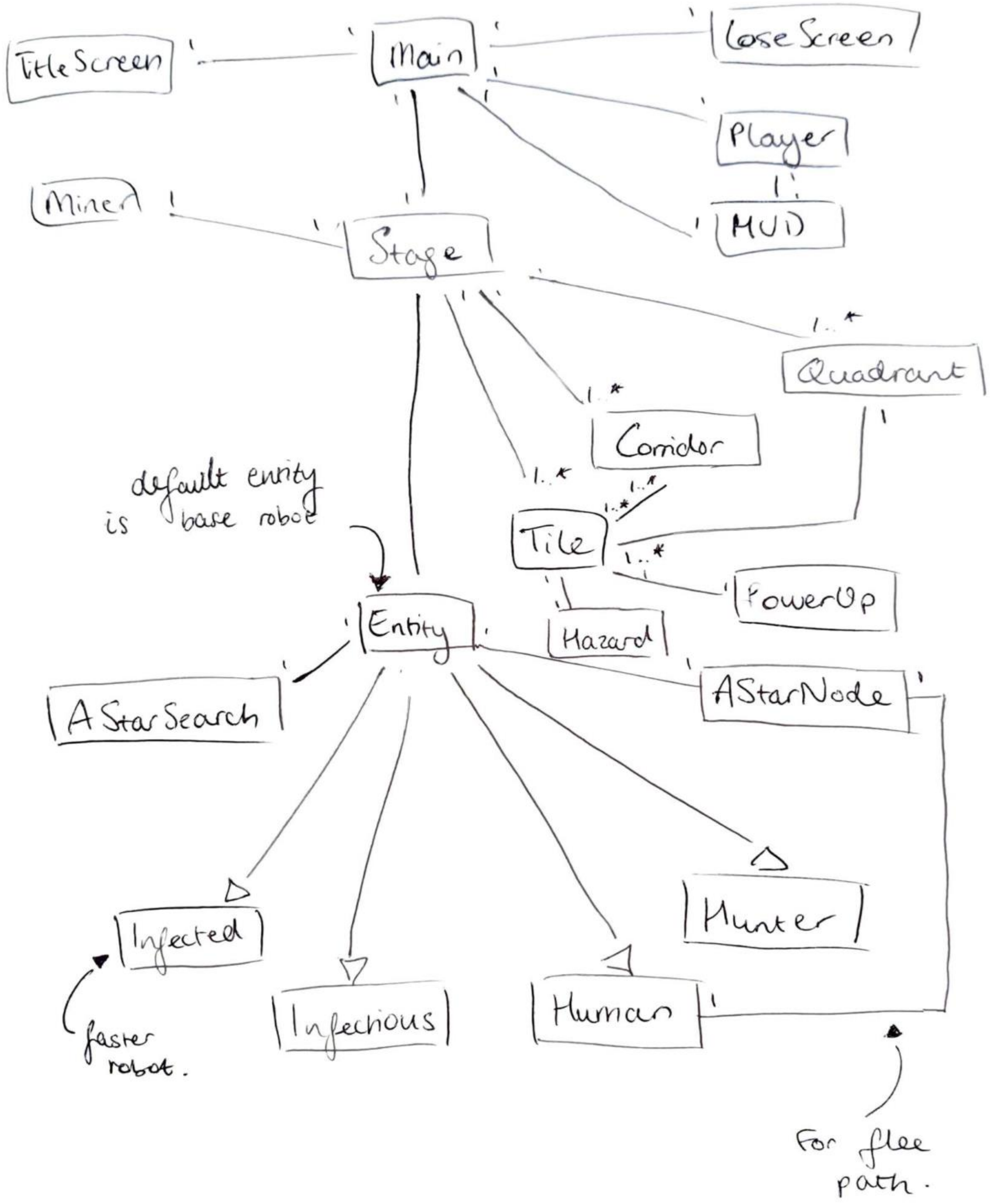


Figure 1 A diagram showing the program's basic class structure.

In the above figure is a diagram providing information on the basic class structure of the game. All non-player characters inherit from the entity class, where the default entity is a basic robot. This reduced the amount of repetitive code in the program, for example, allowing all enemies to use the same searching functions but with different targets. Quadrants and corridors store tiles so they can be visually different in debug mode and become a lighter colour as players explore them. The play area is quite large so visual aids have been implemented, such as highlighting the starting quadrant green and unexplored areas are darker until they are explored.

Player

Controls are handled by determining when buttons are pressed and released, by setting Booleans, to allow for smoother movement and shooting. The player has a cooldown for shooting bullets, preventing them from spamming projectiles to quickly and making the game too easy.

DEBUG Mode

A debug mode can be toggle by toggling the DEBUG boolean in the Main class. This is not necessary to play but will visualise A* search algorithms and colour quadrants and corridors. This was useful for debugging the game and testing that it works as intended. Additionally, in DEBUG mode, pressing space bar will reset the game. Useful information about stage generation is printed to the console.

Procedural Generation

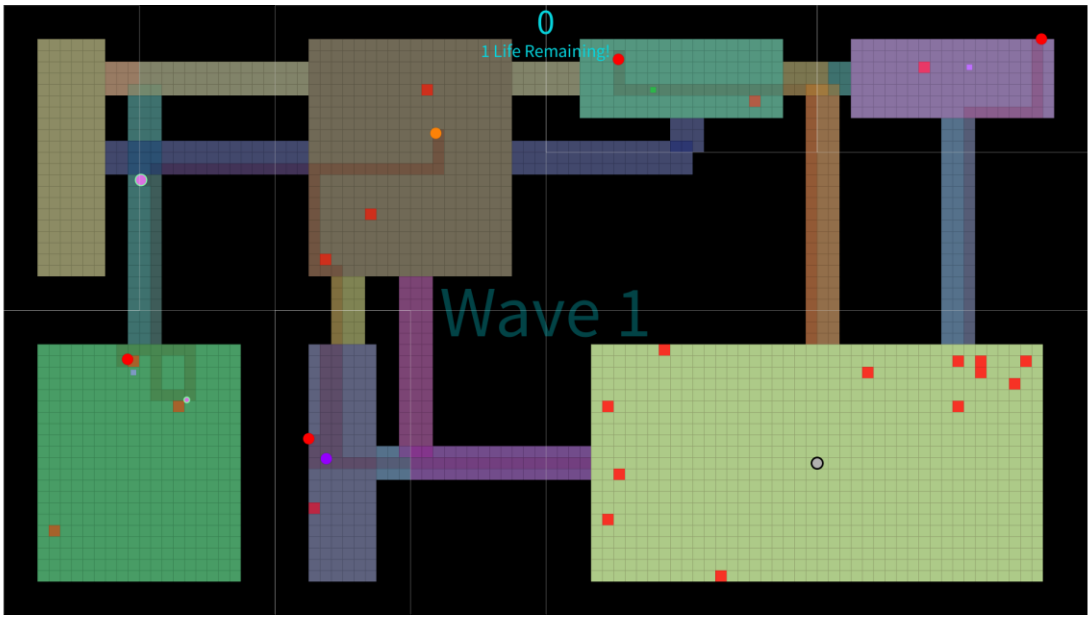


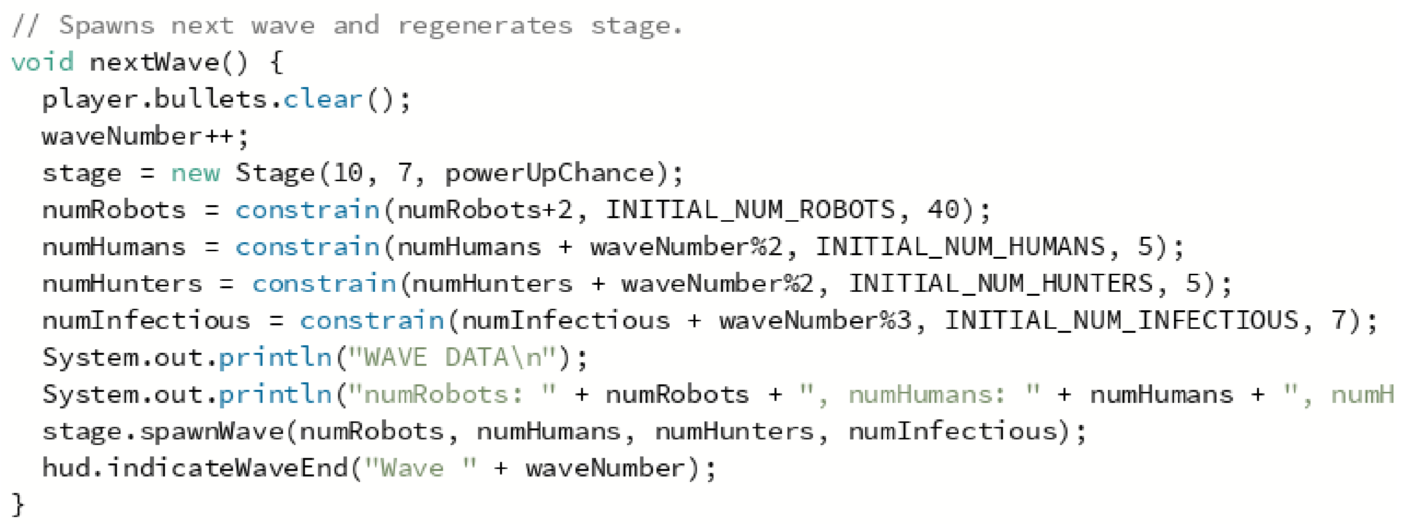
Figure 2 Screenshot of game with DEBUG on.

To generate the rooms, the play area is first split into quadrants which generate rooms using the numSplits and numRooms parameters required to create a stage. Starting off with the first quadrant being the entire play area, quadrants are split numSplits times. To select which quadrant to split, they are ordered from smallest to largest and a random quadrant is picked from the larger half of quadrants. Each quadrant has an even chance of being split horizontally or vertically. From these quadrants, the smallest ones are deleted until reaching the required numRooms. A visualisation of these quadrants can be seen in the above figure with DEBUG toggled on, where the quadrants around the rooms are outlined with a faint white line.

From the quadrants, rooms are generated in the centre of them leaving a margin of 3 tiles. Whilst placing the room in the quadrant, hazards are placed randomly. To connect these rooms, a miner is used to dig between the centres of each quadrant. One quadrant will always have at least one tunnel connecting to another, resulting in a map that is completely traversable by the player and entities. When the player is first spawned in, they are given invincibility to ensure they don’t get hit by hazards straight away.

Each time a new stage is generated each room has a 10% chance to spawn a power up that will either give the player another life or invincibility. The powerups that gives the player another life are small green squares, and the powerups that gives the player invincibility are small purple squares.

Waves and Spawning Enemies



```
// Spawns next wave and regenerates stage.
void nextWave() {
  player.bullets.clear();
  waveNumber++;
  stage = new Stage(10, 7, powerUpChance);
  numRobots = constrain(numRobots+2, INITIAL_NUM_ROBOTS, 40);
  numHumans = constrain(numHumans + waveNumber%2, INITIAL_NUM_HUMANS, 5);
  numHunters = constrain(numHunters + waveNumber%2, INITIAL_NUM_HUNTERS, 5);
  numInfectious = constrain(numInfectious + waveNumber%3, INITIAL_NUM_INFECTIOUS, 7);
  System.out.println("WAVE DATA\n");
  System.out.println("numRobots: " + numRobots + ", numHumans: " + numHumans + ", numH
  stage.spawnWave(numRobots, numHumans, numHunters, numInfectious);
  hud.indicateWaveEnd("Wave " + waveNumber);
}
```

Figure 3 Code snippet demonstrating wave mechanics.

After each wave, a new stage is generated and the number of robots, humans, hunters, and infectious are increased, as shown in the above figure. The difficulty of waves scales with the wave number, until maximum values are reached – only allowing a maximum of 40 robots, 5 humans, 5 hunters, and 7 infectious. The spawning of robots and humans considers the player's spawn quadrant so that they do not get immediately attacked or immediately save a human – robots and humans will be spawned in a random place within a random quadrant, excluding the spawn quadrant.

If the player kills a robot, 1 point is added to the score, whereas if they kill humans significantly more is added – 10 for adult humans, 15 for teens, and 20 for children. This scoring system is intended to condition the player to prioritise saving the humans over robots and play faster.

Robot and Human AI

Robots and humans both inherit from the Entity class, so that they can make use of the same movement functions with different targets. Changing the target of an entity can be done by overwriting the getTargetCoords function in another class. To create different behaviour when an entity reaches its target the onTargetCollision function can be overwritten. For example, a standard entity (robot) will get the player's coordinates for it's target and cause the player to take damage on collision, whereas the hunter will get a human's coordinates and kill the human on collision.

Entities can "see" something when they are in the same room, corridor, or they are less than or equal to 8 tiles away according to an A* search. A* searches are implemented for chasing and fleeing mechanics using code from the tutorials. To chase a target, entities will move towards the next tile in the A* search or move away to flee. Whilst patrolling, entities will slightly change their orientation randomly and continue moving a direction, and if they hit a wall, they will continue to orientate themselves.

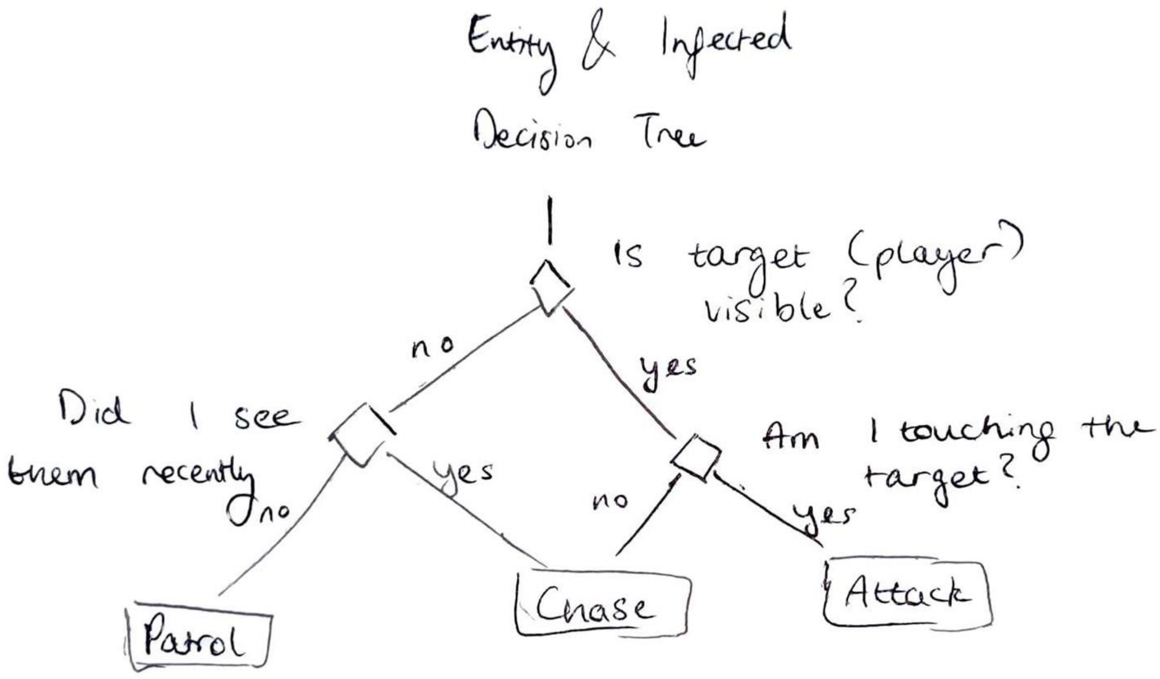


Figure 4 A decision tree for basic robot entities and infected robot entities' behaviour.

Basic robot entities and infected robots make the same decisions for whether they are patrolling, chasing, or attacking the player, as can be seen in the above decision tree figure. They will patrol or chase the player dependent on whether they have seen them recently and when they are close enough, will attack the player.

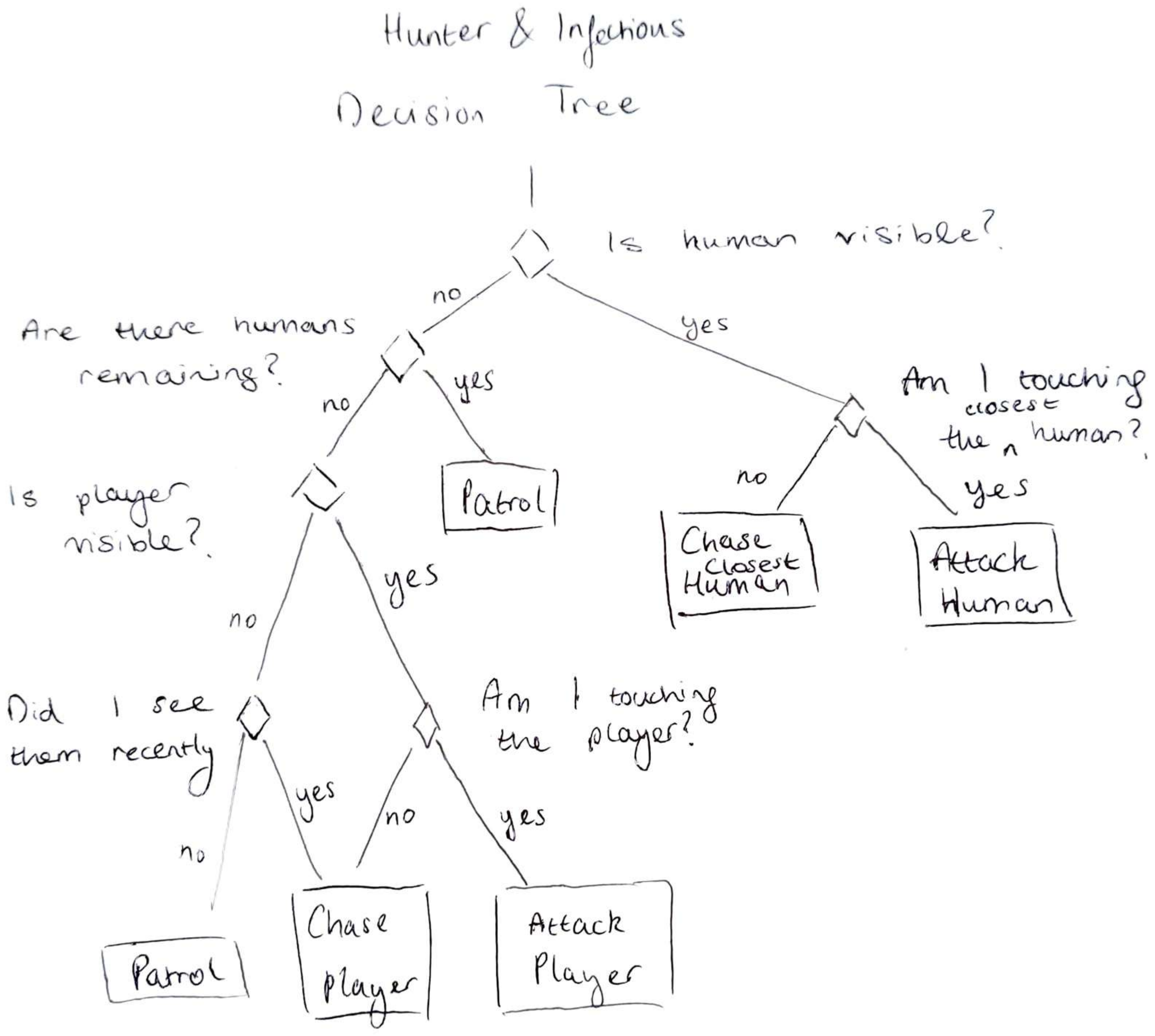


Figure 5 A decision tree for hunter and infectious robot entities' behaviour.

Hunters and infectious robots make the same decisions for whether they are chasing, attacking, who they are chasing or attacking, and whether they are patrolling. If they cannot see the player or humans, they are patrolling. The hunters and infectious robots will prioritise chasing and attacking humans and will ignore the player until all the humans are gone – behaving like regular robot entities.

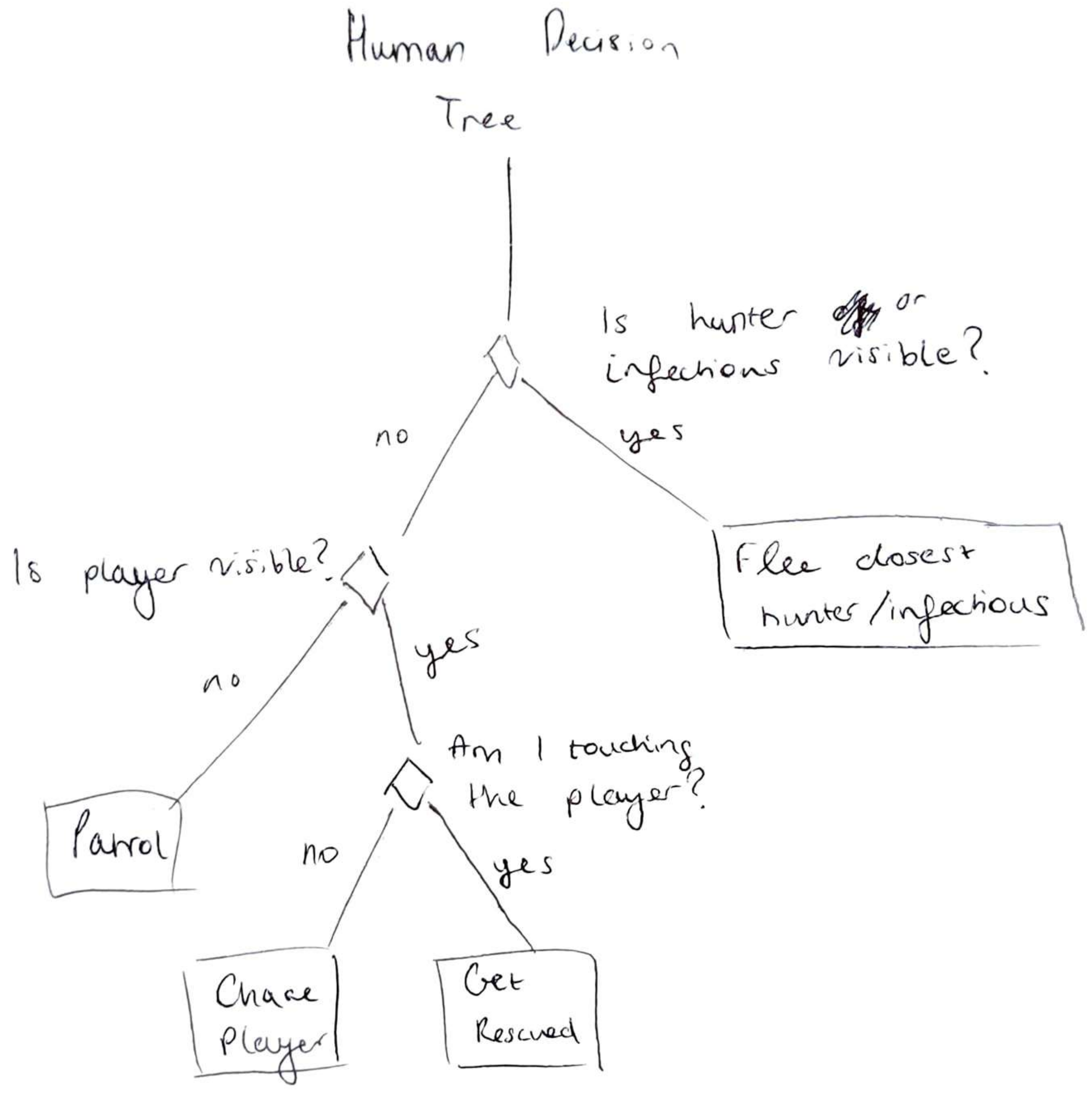

Human Decision Tree
Is hunter or infectious visible?
no
yes
Flee closest hunter/infectious
Is player visible?
no
yes
Patrol
Am I touching the player?
no
yes
Chase Player
Get Rescued

Figure 6 A decision tree for human entities' behaviour.

Human entities will patrol the stage area until they see a player or hunter/infectious robot. If they see a player and cannot see a hunter/infectious robot, they will move towards them to try to get rescued. However, if they can see a hunter or infectious robot, they flee towards the edges of the room to try to stay alive as long as possible.

Testing

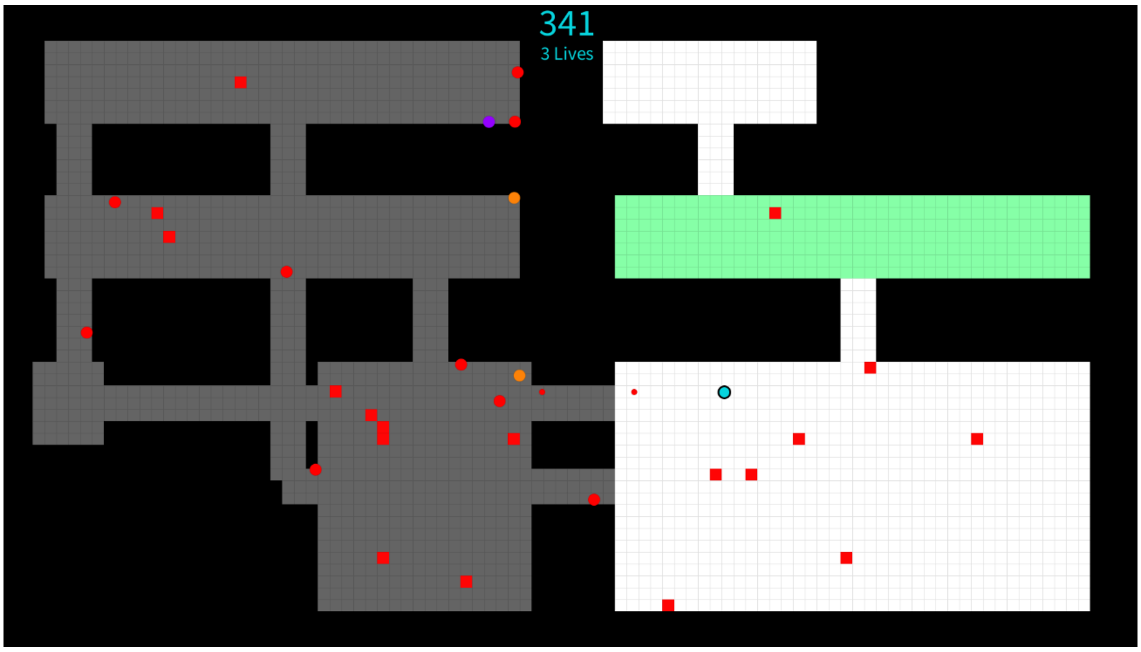


Figure 7 Example screenshot of playtest.

Testing of the game involved numerous playtests, such as the one shown in the above figure. Each playtest involved taking note of what changes needed to be made, either through bugs discovered or changes to difficulty. For example, shot speed has been decreased dramatically from the original version to increase the difficulty of the game. The DEBUG mode also made it easier to test the placing of enemies and procedural generation by repeatedly pressing space bar to reset the game.

Evaluation

All the elements of the specification have been implemented into the game, despite a short development time. The graphics of the game remain relatively simple, as can be seen by the above figure. Given more time, it would benefit the game to add textures to make entities more distinct and descriptive. For example, having different looks for each robot. Additionally, the AI of the entities could be improved so that patrolling entities explore room to room and humans will attempt to flee past hunter/infectious robots to another room and "shake their trail". However, I could not implement more complex AI mechanics within the practical time frame.

Conclusion

Given more time, I would like to make the changes mentioned in my evaluation, such as improving the AI and graphics of the game. However, this is a complete functioning prototype of Robotron with all the entity mechanics, power ups, waves, lives, and scoring system that could be useful for playtesting and tweaking. Entities use decision trees to determine their behaviour and make use of the A* algorithm and kinematic algorithms from lectures.

References

Canva. https://www.canva.com/ (Accessed: 15/03/2024)