Assassin: generated dungeons with hide-and-seek agents

Introduction:

I got the idea from stealth games like *Hitman* and *Assassin's Creed*. The goal of the game is to kill all the targets (red) while avoiding guards (blue) and overseers (yellow). The third-party scripts I used are the agent steering basics, rigid body physics from the labs, and the NPBehave library. Everything else is written from scratch.

At the start of the game scene, the game manager (with script GameManager.cs) calls the dungeon generator (DungeonGenerator.cs) to build the dungeon, then it calls the obstacle spawner (ObstacleSpawner.cs) to add some obstacles in each room of the dungeon. It then calls the unit spawner (UnitSpawner.cs) to place agents in the dungeon. At run time it checks if all the targets are eliminated. If so, the player wins the game. The camera manager (CameraManager.cs) keeps the camera centered on the player.

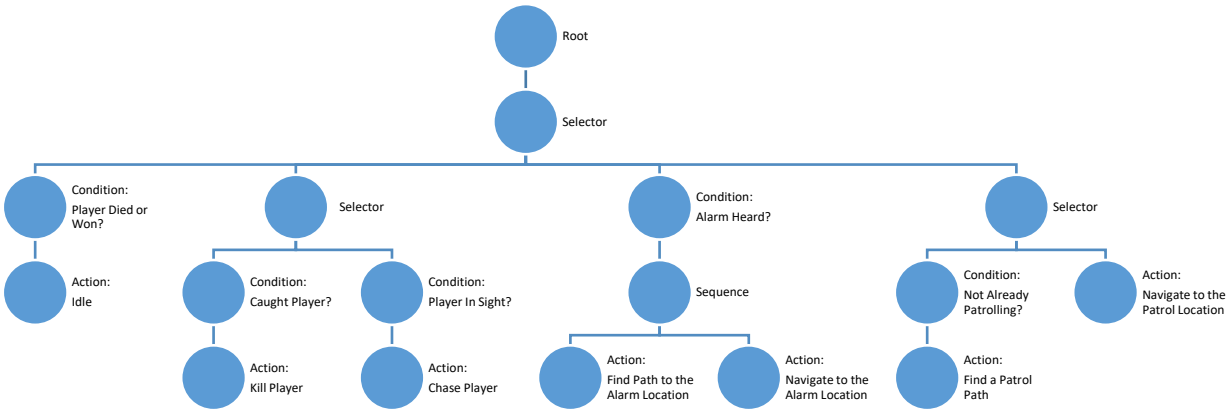
Agent:

Player (PlayerUnit.cs):

The player (green) is a simple unit that listens to the keyboard input. Navigate using the classic "wsad" control.

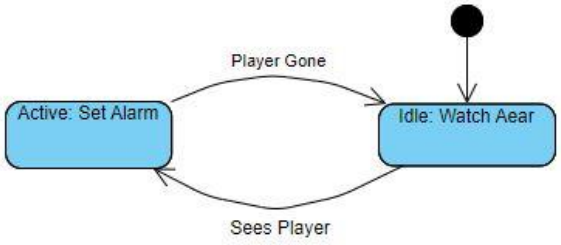
Guard (GuardUnit.cs):

This (blue) is an advanced behavior tree agent. If it catches the player, the player dies and loses the game. It chases the player if it sees the player. Otherwise, it will respond to alarms set by the overseer. If it has nothing to do it will randomly patrol the dungeon using waypoints. The flowing diagram shows the detailed NPBehave tree structure. The UnitEvent.cs implements some methods for checking if a unit catches another, etc.



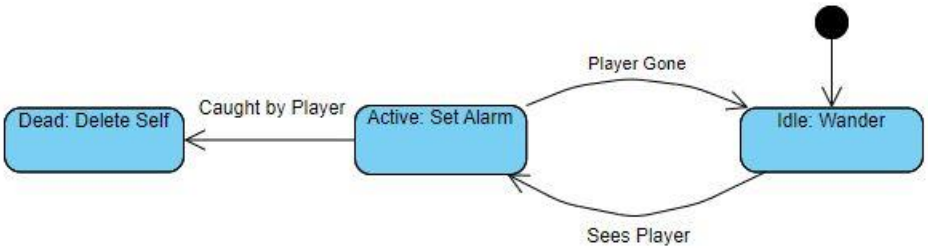
Overseer (OverseerUnit.cs):

The overseer (yellow) is a simple finite state machine (FSM) agent. When it sees the player, it will turn orange and alert all the guards calling them to investigate the player's location. When the player leaves its sight, the guard will go check where the player is last seen by an overseer. This is done by creating a dummy agent (translucent green) at the players last seen position and let the guards navigate to it.



Target (TargetUnit.cs):

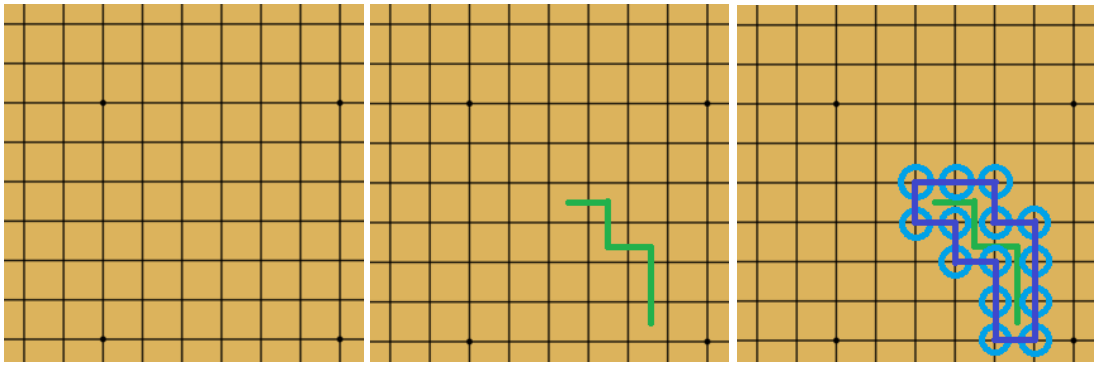
This agent (red) is the goal of the game. It will try to hide from the player when approached. If the player catches it, it will die. It is also implemented as an FSM agent.



Generator:

The dungeon generator I implemented is similar to the one used in the game *Unexplored* where an advanced cyclic dungeon generator produces a map that is made of a big cycle. The algorithm is a bit like cave digging but way more complex. The reason why a cycle maybe good for this game is that in stealth games the player is encouraged to sneak around enemies. If the player can cause some distraction on one side of the map and mislead the guards away (for example, activated an overseer), he/she can go around the other way to reach the target.

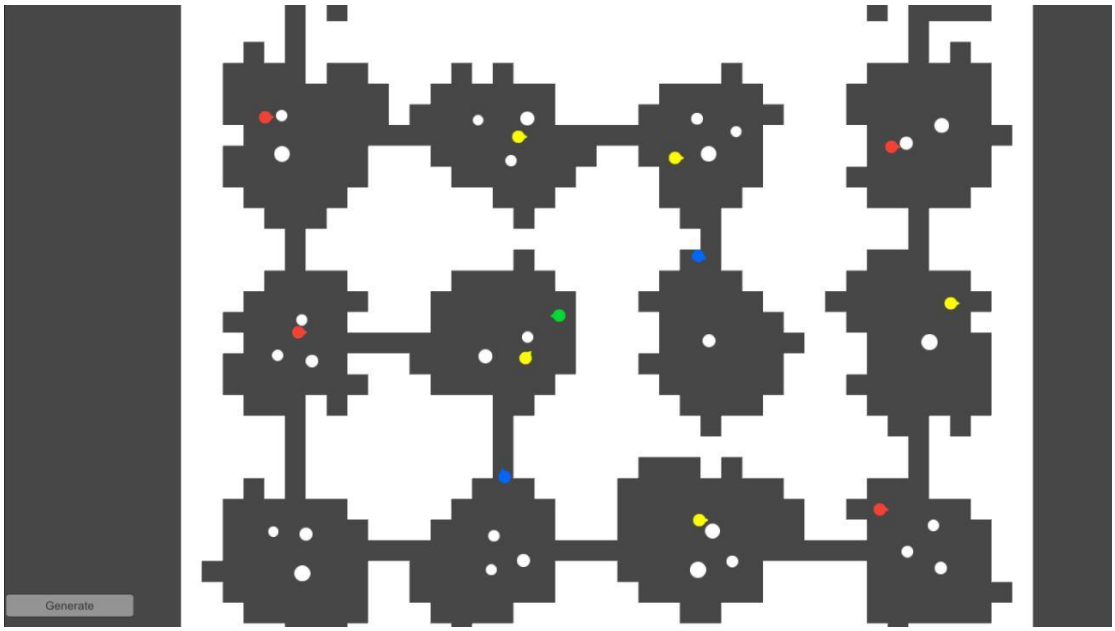
The basic idea is that it starts with a 5x5 grid like a Go board (the size is adjustable), so the cross points are the nodes, and it then performs a random walk on the board on not the cross points but the cells instead. Doing so resulting in a path made of connected cells. Each cell is adjacent to four cross points, so wrapping up the cell path will produce a cycle made of those cross-point nodes. The figure below shows an example of how this is done.



The random walk on the cells (green) generates a cyclic graph (blue nodes and purple edges) on the cross points. During the random walking, random adjacent cells are visited with a certain chance to add some rooms around the main cycle. After that, the grid is expanded to the real size (for example 25x25) with rooms being squares connected with narrow corridors. Then the map is polished to make the boundaries of the rooms look more random. During the dungeon generation, waypoints are also created for agent navigation. As the graph is derived from a 5x5 grid, it is easy to compute a navigation graph by simply pick the center of the rooms and the corridor entrances to be the waypoints.

The obstacle spawner simply places circular obstacles in each room with randomized size and location. It actually uses the waypoints to access the location of each room and avoids putting objects on the waypoints.

The unit spawner also uses the waypoints to access each room. First it places the player in a random room. Then for every room there is a chance to spawn something, either a guard, an overseer, or a target. It is guaranteed to spawn at least one target and one guard. When placing units, the spawner will check the location's availability by asking the obstacle spawner if there are any obstacles spawned there.



A generated example dungeon. See [Link] for a video illustration.