Adventures in PCGPlanworld

Aim of the project

The main goal of the project was to create a demonstration of Procedural Content Generation, Planning and A* Pathfinding. To get to that point, the scenario of an adventure game, with AI controlling all agents, was deemed as suitable.

The world would pit two (or potentially more) factions against each other, each with different goals and ways to achieve said goals. Each agent would think independently, yet be able to ascertain on what allies have done to further the cause, as well as decide on what enemies have done to thwart them.

However, such a scenario would get boring fairly quickly. As a result, randomly generated levels would add to the entertainment of the demonstration, while also offering a great proof of concept should an actual game need to be developed.

For a bit of extra variety, a further stretch goal of having two different scenarios was decided upon. The first one was of a forest, where one faction attempts to steal a treasure, then escape, whereas the opposing faction tries to keep it protected. The second scenario is simpler, with adventurers attempting to escape a dungeon, while the local werewolves want to kill all trespassers.

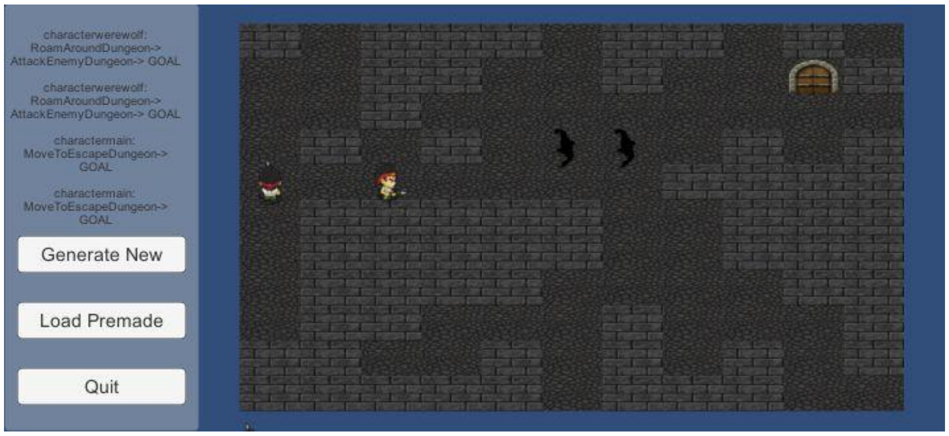
Descriptions of the scenarios

Forest scenario

This scenario pits rogues against spirits in the forest. The rogue wants to steal the treasure, then escape. To do this, they can clear trees in their way (only done as a last resort). They must avoid the killer spirits, which roam mindlessly around the forest. The protector spirits simply plant trees around the treasure to protect it from prying hands.

Dungeon scenario

When adventurers want to escape the dungeon, they must traverse it. On their way, scary werewolves might attempt to stop them. When that happens, they have to fight for their lives. They have health, attack and different attack speeds. After a battle, they have the chance to stop and heal.



```
characterwerewolf:
RoamAroundDungeon->
AttackEnemyDungeon-> GOAL

characterwerewolf:
RoamAroundDungeon->
AttackEnemyDungeon-> GOAL

charactermain:
MoveToEscapeDungeon->
GOAL

charactermain:
MoveToEscapeDungeon->
GOAL
```

Figure 1- Dungeon scenario

Tools and techniques used

Game engine

The chosen game platform was Unity 5, as it offers a lot of great tools for working with sprites and create quick prototypes for ideas. The use of C# is a bonus.

I created sprite animations by using a wide variety of sprite sheets found on Google Images, as well as a nice tutorial from Michael Cummings [1].

Every different tile type is its own prefab, for the purpose of clarifying utility.

Everything scales correctly with various resolutions and screen sizes, as it calculates everything based on camera viewport (a bit of code stolen from myself!).

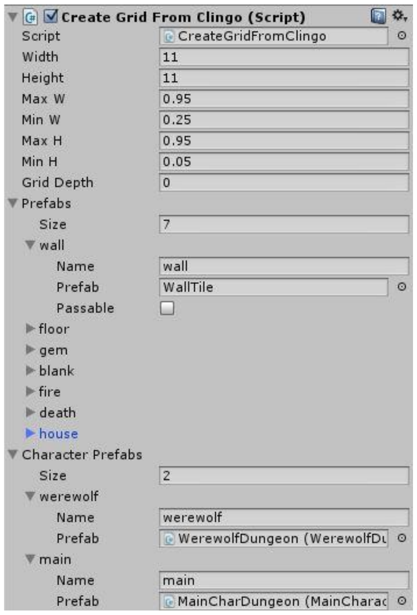
Clingo

For the purpose of creating sensible levels for the action to happen in, the tool and technique used was Clingo [1] with Answer-Set Solving. A couple of level definitions were written (available in Tools/clingo/level.txt and Tools/clingo/levelDungeon.txt) to describe the facts about the world for use with the solver.

Invoking Clingo with one of the definitions would attempt to create a valid solution, which would then be further passed on to the game itself for rendering and playthrough. The most advanced bit of code is making sure there is a potential path between the adventurers and the exit in the dungeon scenario. The forest scenario has a similar check, but less forced, as the rogue might chop trees down.

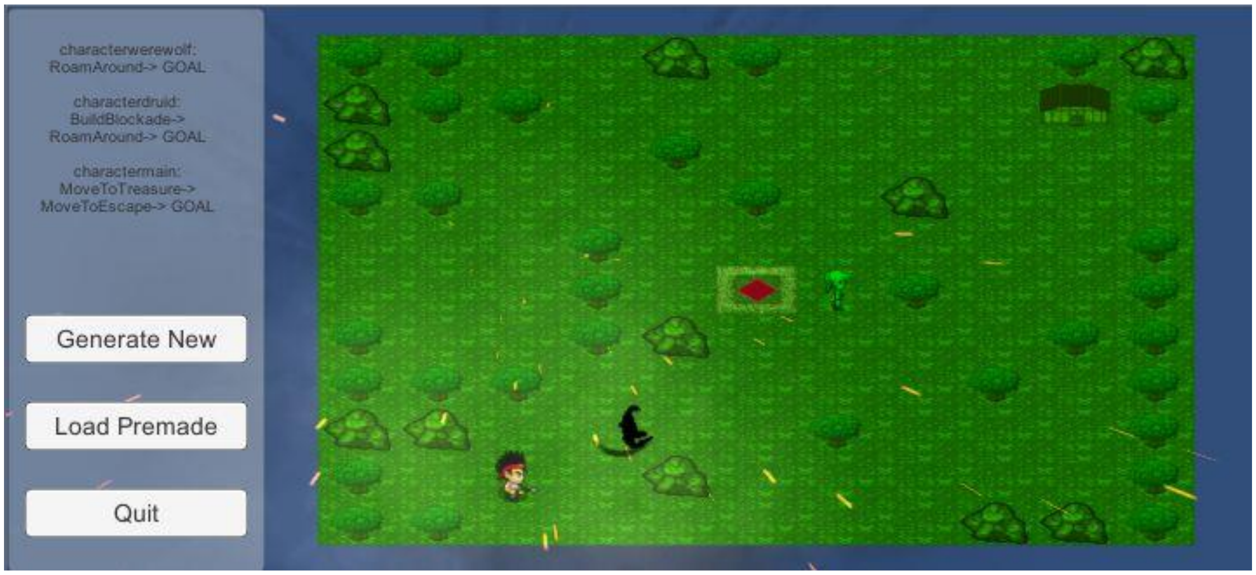
To interact with Unity, I made a script that would invoke Clingo, pass the desired level definition to use, then retrieve the generated level design and store it in a Unity-friendly fashion, as well as render it. This script allows users to define whatever type of sprites they want, as it offers an Inspector extension for them to populate with their own prefabs, suitable for anything they need. (See Figure 2)

Clingo would then generate sensible looking levels, while the script would make them look presentable. (See Figure 3)



```
▼ Create Grid From Clingo (Script)
  Script            CreateGridFromClingo
  Width             11
  Height            11
  Max W             0.95
  Min W             0.25
  Max H             0.95
  Min H             0.05
  Grid Depth        0
▼ Prefabs
    Size            7
  ▼ wall
      Name          wall
      Prefab        WallTile
      Passable      ☐
  ▶ floor
  ▶ gem
  ▶ blank
  ▶ fire
  ▶ death
  ▶ house
▼ Character Prefabs
    Size            2
  ▼ werewolf
      Name          werewolf
      Prefab        WerewolfDungeon (WerewolfDu
  ▼ main
      Name          main
      Prefab        MainCharDungeon (MainCharac
```

Figure 2 - Clingo Loading Script


characterwerewolf:
RoamAround-> GOAL

characterdruid:
BuildBlockade->
RoamAround-> GOAL

charactermain:
MoveToTreasure->
MoveToEscape-> GOAL

Figure 3 - Resulting rendering

Goal Oriented Action Planning

While the process of generating interesting levels is great, it doesn't mean much without something happening on the screen. While the initial idea was to have an interactive experience, that would have required a lot more design time, as well as mechanic testing. As a result, completely AI-controlled agents were the way to go. And to satisfy a level of simplicity, yet scalability, (as well as unfamiliarity with the technology) GOAP was the chosen technology.

The planning library used was an open source one done by Brent Owens [2]. This implementation offered a minimal solution to the planning tasks required, as it is barebones, yet functional.

Given a world state (computed independently for each agent) and a list of actions they could do, each action having requirements and effects, the planner would compute a series of actions getting said agents from their current state to a final, more desired state.

One of the greatest hurdles for the task at hand with the provided code was the fact the computed plan could not be interrupted mid-movement, should the requirement for it come up. I did not want an agent to keep walking towards a treasure that was no longer there, or to just walk through a werewolf. That would have been dumb and beside the point of having an AI. The way I worked around this problem, without changing the original code, was to have each action include movement in its performing of the action, then aborting as soon as the environment changed, forcing the generation of a new plan.

The result of this minor 'hack' was agents that reacted quickly to their environment, creating appropriate responses and behaving in a smart fashion.

What I would do to further improve this would be to change how the planner looks at the world state, aborting the current plan as soon as that state changes in a manner different from what was predicted (as a result of its actions).

During the development of the planner, I was confused as to its usage. Preconditions and performing and so on were confusing. I decided to read Jeff Orkin's article on it [4], to hopefully understand it better. It did a good job of explaining the whole process, as well as clarify why my movement hack was, for all intents and purposes, not a bad idea. Not optimal, but a step in the right direction. Constant replanning is a critical part of GOAP being a good tool for AI development.

I believe GOAP to be a good choice for the demonstration, even though the amount of actions is not great. It offers scalability and stability, while keeping everything interesting and smart.

A* Pathfinding

This is a very simple algorithm for finding short paths within a game world. It is used to figure paths from a tile to another, as well as decide on whether paths even exist. Not much can be said about it, as it is very standard and easy to integrate, as well as poses no real technological complexity in integrating it, beyond defining what makes a valid neighbor to a given location.

An interesting trick done to dissuade a rogue from going next to werewolves is to increase the cost of walking around a werewolf. That way, paths avoiding the werewolf are preferred, even if they are longer.

The presence of constant replanning results in agents not wasting time with their current path if it is suddenly blocked (by a green spirit for example).

Potential for the future

There are many things that could be improved. More actions could be added to the planner. The complexity of the levels could be increased. A simple narrative could be done whenever an agent changes their plan.

The one thing that I would truly like is to directly integrate a Clingo-like solver into the project, to bypass the need for external applications. This could result in a web-based version of this project!

Of course, the biggest potential upgrade would be to the basic GOAP implementation, to add support for the state-based replanning. That would be cool.

Conclusions

This was a very educational exercise. I had never played around with GOAP, while PCG was also a topic I dabbled in very little (some Perlin Noise generation several years ago, but nothing more). Learning a bit more about declarative languages was also interesting.

I would happily use a more robust implementation of GOAP in the future, maybe even write my own should the time and project allow it.

A good outcome of this project is, I believe, my Clingo-output rendering script. It is flexible, it works well and does no further processing of the Clingo definition, thus being quite fast. It also takes blank, undefined spaces in consideration as well, which is a plus!

For 2 and a half days of work, I'd say this was a great success!

Links

GitHub repository: https://github.com/mihail-morosan/IGGILondonMarch

Video: http://youtu.be/-Q1H0gASyLc

Download: http://morosanmihail.com/home/files/IGGI/LondonIGGI2015-Mihail.zip

References

[1] "Creating 2D animated sprites using Unity 4.3," [Online]. Available: http://michaelcummings.net/mathoms/creating-2d-animated-sprites-using-unity-4.3.

[2] "Clingo," [Online]. Available: http://sourceforge.net/projects/potassco/files/clingo/.

[3] "GOAP," [Online]. Available: https://github.com/sploreg/goap.

[4] J. Orkin, "Three States and a Plan: The A.I. of F.E.A.R.," 2006. [Online]. Available: http://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.