**COMP 452 TME 3**

**Jason Bell 3078931**

**June 26, 2016**

**Overview**

The game executable can be found in the /dist folder

The sprites were drawn myself, and are clear evidence of my complete lack of artistic talent.

The only sound file is the “Wilhelm Scream”, which has no owner and is so ubiquitous to be in the common domain ( <https://en.wikipedia.org/wiki/File:Wilhelm_Scream.ogg> )

**Implementation**

This assignment was implemented in Python, with permission from Dr. Esmahi, to gain experience with that language: I already have extensive experience with Java. He required that I provide both source and built executables to ease marking.

I found this assignment quite challenging. In fact, after finding the first two assignments quite simple, I overestimated how much time this one would take, so am submitting it a day late. Entirely my fault of course.

While implementing the minimax and alpha-beta pruning algorithms for the connect 4 game was quite straight-forward, I found it quite challenging to determine how to get the AI to play well: it was a “garbage in, garbage out” situation. That is, it doesn’t matter if the minimax algorithm is implemented correctly if you’re not giving it useful data from which to give a decision. After trial and error, I found a set of scores to assign three different states that appears to work quite well:

1. 4-in-a-row: a win, basically infinite score
2. 3 chips of same color, and one empty spot: high score, as this is a potential 4-in-a-row
3. 2 chips of same color, and two empty spots: lower score, as this may potentially lead to #2

Thus far I have been unable to beat the AI, though I’m unsure if this is a sign of good AI or my own ineptitude at playing connect 4.

For game 2, implementing ID3 was challenging mainly in that it’s a very complex algorithm: I spent a lot of time chasing after bugs. The most embarrassing issue was that I started getting frustrated over the number of times that ants would generate an error state when making a decision. After some investigating, I realized that this was simply because the ant didn’t have an example of how to handle its current state. I adjusted the code so that when an error occurred, I could view the ant’s state and add an appropriate example. Nice bit of learning in action there.

Game 2 is the “ants” game from TME 2 re-implemented to use an ID3-built decision tree rather than a finite state machine. Behavior of ants is similar to those in TME3, except that they will remember where water and food is. If they have that knowledge, they will path to one of them rather than wander. One interesting example of “emergent behavior” is that once food is found that can be reached from home without crossing a poison tile, the population explodes. In fact, in TME2 I used a low number of poison tiles, as ants wandered into them too often and the population crashed. For TME3 I increased the number of poison tiles to help cull the ant population. Still, when testing this game, I recommend starting with 10 or more ants: the ants still start out wandering, and it seems like most of the time a single ant inevitably wanders into poison before finding food.

Game 2 uses the same A\* pathfinder from TME 2. This pathfinder is interruptible and generates the path in steps so that it’s progress can be viewed. This also means that it could be invoked across multiple frames.

Game2 doesn’t fit the assignments definition of a “game” as being played against an opponent with a winning state. However, I feel that requirement is too constraining: a learning algorithm is still demonstrated, and implementing and watching it certainly feels like a game in the vein of simlife, simcity, etc. A game doesn’t have to have an ending state to be fun.

If you wish to run from the scripts, you will need to install 32-bit Python 2.7.

Then, install any version of Visual Studio 2015. Full versions are available through DreamSpark, and the community and express versions will also work. You’ll need to install python tools for visual studio: you’ll see an option to do this in the “new project” window.

Once python tools have been installed and this project is opened, you may need to install pygame. To do this, you should see a “Python Environments” tab in the upper right next to the solution explorer. Click it, then select Python 2.7. Use the pip option to search for and install pygame (pip is Python’s equivalent to Maven).

I do not recommend trying to use py2exe to build an executable: I suggest removing BuildExe.py from the project. Then, start the game with Game1.py or Game2.py as the startup script.