Coursework 3: Ethical Decisions in Combat Games

Joanna J. Bryson and Andreas Theodorou

March 22, 2018

1 Introduction

Your final coursework requires building AI with two essentially human-like components: first, having multiple, conflicting goals, with mutually-exclusive methods of fulfilling those goals; and second that some of these goals are social while others individual. Social goals cannot be pursued without individual welfare, and more generally mundane goals can become urgent for viability if they have been too long neglected.

This coursework offers an option for a very light-weight writeup, plus points contributed from participating in a games competition with your classmates. Alternatively, you can write a two-page report like your previous courseworks.

2 Methods

Capture the flag (CTF) is a popular multiplayer mode in first-person shooter games, where the participants split into teams. Each team has a flag located in its *base*, which acts as its starting location. The objective of the game is to capture the other team's flag located at the team's base and safely take it to their own base. Players may combat and 'kill' enemies, or avoid them altogether. Killed enemies often respawn at their original base after a set time, however multiple varieties of the game exist with different rules for respawning and scoring.

In CW3, a game consists of three four-minute rounds between two five-person teams, and killed players won't respawn until the end of a round. Also, a team can score even if the enemy currently holds their flag. A team can therefore achieve a high score in an individual round by simply exterminating all of its enemies and scoring the flag multiple times. However, if the surviving team fails to score, the eliminated team is awarded a point. The winner of the match is the team with the highest combined score for the three rounds. However note that tied standings within a tournament will be resolved by highest total point scores, so it may advantage both teams to play for high score rather than elimination.

Each team is invited to take part in a tournament. First, a qualifier stage will take place. In this stage, groups of four teams will be formed. Each team is expected to compete against all teams in its group, scoring 2 points for each victory or 1 for each tie. In addition, overall point totals will be kept to resolve any ties in outcomes. The winner from each group will advance to next stage; a round-robin league.

3 Results

If you choose to take part in the tournament, your grade will be 30% determined by your code and its performance, and 70% on a brief report. Regardless of your team's performance, assuming sufficient work was done (see next paragraph), you will receive 20% for participating in the tournament. The remaining 10% of implementation marks will be awarded based on your performance at the tournament; 5% for passing the qualifier stage and a further 5% for winning the tournament. **Note: your code must run on the laboratory machines in tournament mode at the beginning of the competition.** Be certain you have practiced a tournament with someone else's bots beforehand; you can do this in a previous lab.

In the mandatory tournament lab you will first show the tutors how to run your code, and show them how your code is different from the distributed code. **If you cannot run your code and show how you modified it, you will not get the 'free' 20% of the mark.** If you cannot attend your own lab, you can—*with an email record of consent of the tutors*—either have another student do the demo for you, or attend another lab. *Be sure that you have been CCed on an email where the TAs notify the lecturer of your permission to attend another lab, xor have someone else run your code.*

The majority of your mark (up to 70%) will derive from (up to) 10 brief written observations; these should take approximately one side of paper, Times 12 point font. If you go over by a line or two, dont spend a lot of time fiddling with margins. But going substantially over a page will be penalised in proportion to the extra length, e.g. two pages is the same as zero pages.

Do not feel obliged to recount exactly how your bot did in the competition--this information will come from the tutors. Rather, an observation should be a point about what worked or what didnt work with respect to your task. Observations can be about human-like cognitive systems more generally, or cooperation more specifically. They can be informed critiques of the software tools. An observation can be of more than one sentence, so you might want to start with one sentence as a hypothesis, and then add one or two sentences describing evidence that lead you to believe it, or an experiment about how you would test it. In general, while your own bots may serve as inspiration, you will get more points if you draw conclusions that might be applicable more generally, or that you can relate to themes from the course.

The observations will be marked on a three-point scale:

- 0: Missing or redundant.
- 1: Conspicuously inaccurate or not entirely coherent, but not entirely wrong.
- 2: A good solid observation.
- 3: An exceptionally insightful point.

If you do not participate in the tournament, you can write a two-page report in the general format of the first two courseworks describing an experiment conducted using the system for 100% of your CW3 mark. This coursework as a whole is worth 15% of your unit mark.

You must submit a Unity prefub containing your team. The prefab should be named ⟨*your Bath username*⟩*_team.prefab*. The prefab should contain the *TeamMembers* script; each element of the *TeamMembers* array must have the prefab of an agent assigned to it. Each agent's prefub should have the *AIModule* script, with the relevant *plan file, BehaviourLibrary,* and *NavMeshController* files assigned to its variables. Your final code submission should contain all files required for your team to work, compressed in a *Zip* file named ⟨*your Bath username*⟩*_cw3.zip*. Your final report submission should be in *.PDF* format. The report but not the code can be anonymous. *Note that these are two separate submissions.*

Appendix: The BoD UNity Game (BUNG)

The game, BoD UNity Game (BUNG), takes cues from Unreal Tournament 2004 and other shooters. It is a team-based capture the flag developed in the popular games engine Unity. The human players act as spectators, watching the two teams of five bots as they fight against each other.

The game contains two camera views; the first is a God's eye view and the other follows a single agent from a third-person view. The former allows the user to move and inspect the game from a top-down view. This perspective allows the spectators to observe how different agents interact with each other, helping them study their behaviour. The second camera option allows the spectator to follow through a third-person perspective an agent of their choice, helping to focus on a particular agent. This view should be useful for developers working to monitor and debug specific behaviours.

The game comes as a Unity project. The project contains the following folders:

1. AgentsTeam: Contains sample agents and teams as prefabs.
2. AIModule: Contains the planner. You should not modify assets located in this folder.
3. GameBackend: Contains all the backend related assets, including graphics and code, needed by the game to run. You are allowed to explore the folder, but not to modify any assets.
4. Scenes: Contains different levels.

Your task is to work with the code provided to extend the agents' behaviours. The agents are human-like characters. You can control their legs, arms, and head interdependently from each other. You can, for example, look behind while walking forwards. Be careful, the agents will only detect enemies and flags within their field of view.

You are allowed to create multiple copies of and modify the following two classes:

1. StudentsBehaviourLibrary: Extends the *BasicBehaviourLibrary* class, which in turn extends the *BehaviourLibraryLinker* class. The BehaviorLiveraryLinker provides access to the different available sensors, e.g.: vision, and actuators, e.g.: hands to trigger weapon, that the agent has. Your behaviour library must only use properties made available in this class and must not directly extend or have access to the *Agent* class.
2. NavpathController: Provides access to pathfinding algorithms used by navigation-related actions. Currently, an A* pathfinding algorithm is included.

Moreover, you should create your own POSH plans; tools and how-to are explained on section 3 of this document. You are allowed to use third-party libraries, as long as you include them in your submission and do not break any of the rules stated above, e.g. if you give them access to the Agent class, you will be disqualified. It will be your responsibility to download them and install them at the day of the tournament.

Each agent in your team can have its own plan, behaviour library, and NavpathController. We have created a set of *prefabs*; a Unity asset type that allows you to store a GameObject object complete with components and properties. A prefab acts as a template from which you can create new object instances. To ease the prototyping of your agents, we create a *SampleAgent.prefab*, which you should make copies of. In Unity you should be able to drag and link your plan, behaviour library, and NavpathController as shown in the provided example. You are also provided with a *SampleTeam.prefab* which contains a sample, mixed-strategy team. You should make a copy of this, per our submission guidelines 3, and assign agents to it. Once your team is assembled, you can test it by dragging its prefab to the *SubmissionsHere* gameobject found in your project hierarchy. A practical demonstration of this will be provided in lecture.

ABOD3

Plans are written in XML files, with ABOD3 providing a graphical representation and editing. Moreover, ABOD3 is designed to allow not only the development of reactive plans, but also the debugging of such plans in real time. This should reduce the time required to develop an agent.

Once the game starts running, it will automatically connect to an instance of ABOD3, set in debug mode, through TCP/IP. The planner reports the execution and status of every plan element in real time, allowing developers to implicitly capture the reasoning process within the agent that gives rise to its behaviour. The planner has the ability to report its activity as it runs, by means of callback functions to a monitor class.

ABOD3 will always display the plan of the agent currently—or most recently—selected in spectator's mode. This allows a developer to select within the game which agent to debug. To enter spectator's mode, press the "space" key. You can enter debug mode by pressing the *Start U-POSH Server* button under the *Debugger* menu.

Possible Approaches

BUNG is designed to allow a variety of strategies at both individual agent and team levels. For example, at team level, you can approach the challenge in any of the following ways:

1. The whole team rushing to secure the enemy flag.
2. Part or all of the team staying in the base to defend the flag.
3. Search for and eliminate all enemies, ignoring the flag until this is completed.
4. Some combination of the above.

Each agent can be individually customised with its own set of goals and behaviours to satisfy such goals. You need to think of the overall strategy of your team before you assign goals to individual members. Even if two or more members share the same list of goals, they may have different priorities. For example one or more members of the team may prioritise a goal to "Survive" over "Capture the Flag". You need to consider how *altruistic* or not an agent should behave. Should all members of your team prioritise the team's strategy, whichever that is, or should some (or all) prioritise their own survival, or their team mates'? Should they detect and respond to the performance of their opponents, or their own performance relative to the other agents on their team? Remember that intelligence is the capacity to respond to opportunities.