X-pilot Technical Overview
Group 7

Zino Kader, Viktor Barr, David Grahn

Table of contents

1. File structure

```
├── README.md
├── attacking [contains aiming and attacking logic]
│   └── attacker.py
├── excercises [contains introductory-phase excercises]
│   ├── exc1.py
│   ├── exc10.py
│   ├── exc2.py
│   ├── exc3.py
│   ├── exc4.py
│   ├── exc5.py
│   ├── exc6.py
│   ├── exc7.py
│   ├── exc8.py
│   └── exc9.py
├── helpfunctions.py [general help functions for calculating misc.]
├── instructionhandler.py [reads and interprets instructions and delegates them]
├── maps [contains custom maps for testing]
│   ├── obstacle1.xp
│   ├── obstacle2.xp
│   └── obstacle3.xp
├── multiguy.py [main single bot file]
├── multiguystatemachine.py [state machine for our bot]
├── pathfinding [contains A* implementation and helper files for pathfinding]
│   ├── implementation.py [main A* implementation file with algorithms and calculations]
│   ├── maphelper.py [map conversion algorithms, acts as an interface for pathfinding]
│   └── navigator.py [navigating logic, handling and decision-making on the paths given by A*]
├── states.py [state handler for the state machine]
├── teacherbot.py
```

2. Features of the bot

Our bot has a lot of features necessary to play X-Pilot for you. It uses advanced algorithms to find the quickest path from point A to point B. It uses some rad mathematics and physics to shoot moving targets, and it can also pick up desired items. The algorithm used for pathfinding is the A* algorithm which divides the map to blocks where the blocks with obstacles are marked and avoided while making the path. Our bot is able to aim and shoot at moving targets by using the algorithm used in PlayTechs[1] algorithms for aiming a projectile. The bot is also capable of making decisions on its own using a custom made state machine implementation. All the while, the bot will avoid walls whenever it gets too close, no matter what its current mission or state is. Communication can be received from either a teacherbot with designated missions, or the owner of the bot with custom missions.

2.1 Communication

Because there's a chat function implemented in the game many of our bots features are triggered by commands in the game chat. We always place every single message that is directed to our bot, matches our grammar and identifies as an instruction or question in a stack, which makes it possible to register more than one message per tick. We then iterate over the stack and as we complete a task or answer a question; we remove it from the stack, send a complete message, and start handling the next message. That way our bot can keep track of many instructions at the same time and complete them consequentially. Our bot supports questions regarding player coordinates, heading, state, and so on. Almost all of our bots features can be triggered using messages from players or other bots.

2.2 A*

We find the prefered path by iterating over the map blocks, identifying which blocks that are possibly to travel through i.e. which blocks that are walls, and which blocks that are empty void. We then send the map information to the pathfinder which "attaches a score" to all the empty blocks. The block score (h) is the sum of the distance from the bots position to the block (f) and the distance from the block to the goal (g). Consequently, if we always choose the block with the lowest score we travel to the blocks that seem to be leading closer to the goal. Currently, our path-finding algorithm will not choose diagonal paths, even though it can and has been implemented previously, our testing found that the bot crashes into the corners of blocks in the diagonal path. This bug still stands, and that is why we decided to leave diagonal pathfinding out of the final product. Furthermore, the bot has a fixed, low speed when pathfinding as the

[1] From Playtech: the math behind aiming a projectile at a moving target. URL: http://playtechs.blogspot.se/2007/04/aiming-at-moving-target.html

state machine wasn't ready when we implemented pathfinding, which made it impossible for the bot to avoid walls actively while following a predefined path. In other words, it would constantly go back to the blocks that it had avoided when the bot was pathfinding with high cruising speed. The pathfinding cruising speed is therefore fixed in the final product.

2.3 Aiming at moving targets

We calculate where to aim by multiplying the time of impact for the bullet with the relative velocity of the target and the targets relative position. The time of impact is calculated with the code provided by PlayTech but obviously changed to Python syntax. Because of the fact that the relative position and velocity are vectors we use trigonometrical calculations to add them together.

2.4 Asteroids and map-wrapping

Our bot can shoot at moving targets but it can also move on its own. What is the best way to deal with a asteroid swarm on a minimal budget of fuel? We used to same targeting algorithm as from aiming at moving, which predicts where an object will be in the future based on its movement. The best way to deal with asteroids is to stay completely still and allow the asteroids to come towards you, thus saving fuel. Our bot used to use the radar function to locate the asteroids and with the help of prediction algorithm and pythagoras theorem it could calculate which asteroid was closer. There was however a slight problem where the bot would travel across the map borders, thus altering the perception of the x-axis and y-axis. This was solved by allowing the bot to adjust its position and aiming relative to how far away it was from the border, thus allowing us to shoot through borders.

2.5 Finding and avoiding walls

However not all maps are borderless, some are even mazes. For a bot to navigate effectively it has to be able to detect and avoid walls. Our bot managed to do this with the help of builtin functions which allowed the bot to constantly monitor the distance to a wall from its front. The bot also brakes and constantly adjusts its speed to minimize the chance of it hitting a wall by mistake. Our bot used four modes breakwall, thrust, ready and escapewall. Ready is when the bot is waiting for instructions. Breakwall is if it discovers a wall a making the but brake and turn 180 degrees away from said wall. This is when escapewall kicks in, which makes the bot accelerate up to a certain speed from the wall until it finds a target or a new wall.

2.6 Items and targeting

Items are a big part of x-pilot and it allows the player to gain an advantage over his or hers adversaries. It is therefore natural that our should be able to use them. We used our playtech algorithm for predicting where the items would be and then moving there to pick them up. There are a number of different items. All items which the bot can use on itself such as shields and camouflage are quite simple to use from programming standpoint. The tricky items are missiles, lasers etc. all the weapons which targets enemy players. This requires the bot to both use a targeting function, then compare this target to the server id and then aim towards said target and attacking it. However it better for the bot to attack the target which is closest since said target is the highest probability to be the most serious threat.

3. States

The features of our bots isn’t just triggered with chat commands, the state of our bot is determined by the current conditions. For instance, if the bot is commanded to travel somewhere and attack a target, but that target can’t be found, or it wasn’t provided with a valid target in the first place, it will automatically switch to a state in which it attacks the closest targets instead. The structure and logic is all there for more states to be added and more states were planned but time caught up to us. The state machine does not only handle decisions on a higher-level, but in its own has made the code a lot cleaner just by the nature of the architechture of a state machine. By moving calculation logic and abstracting code that is hard on the eye behind several layers, in the state-machine layer *[multiguystatemachine.py]*, you get a quick overview of what conditions that decides a different path of decisions and how these tasks are delegated to different files.

4. External resources

- From Playtech: the math behind aiming a projectile at a moving target. URL: http://playtechs.blogspot.se/2007/04/aiming-at-moving-target.html
- From Red Blob Games: Introduction and implementation of A*. URL: http://www.redblobgames.com/pathfinding/

5. Final reflections

Most problems we encountered were related to poor planning and time estimation. For example, we spent too much time on the first exercises on the main phase, because we wanted to accomplish them in certain ways, and when we had problems we got stuck on them for long periods of times instead of moving on and perhaps fix the problem when we have a single bot with the final file structure.

We feel that we did succeed in the project, everyone did their best by spending a lot of time contributing with ideas for new angles to attack problems from and sharing their perspectives to try to get a more understandable and effective code. We did not quite finish all the exercises on the main phase which disappointed all of us because we hoped for some time to think outside the box by adding features that weren't in the requirements.

We believe that the most valuable thing we learned and what the next years students should think about was that you should organize the code from the very beginning and always keep in mind that the final project should be a single bot. When we started to fuse together all the features we created during the work on the exercises we found it really difficult to refactor the code and put the bot together in a single working instace. This was due to our main philosophy being that we were finished once the code was working, quality, readability and abstraction was not a primary focus. We recommend next-year students to right from the start create a state-machine separate from the bot stub that communicates with files in different abstraction layers. This will allow you to work on the same files for each excercise, and have the single bot readily available when the excercises are completed.

Primarily we recommend that if you don't have any experience with programming or have had a hard time with the other python courses you should not choose this project. Since the project deadlines have been postponed both the current and previous year, and that the groups from other projects have had time over to improvise and add voluntary features to their project, it shows that this project has much higher demands and requires more time than some of the other projects. However, because of the high requirements you learn a lot from this project if you can keep up. Most importantly, having fun and writing fun code should be a high priority. Fun code is somewhat vague, but what that means in essence is that if the group decides on some simple rules and standards of abstractions and code quality, the code can be more accessible for everybody, thus letting you quickly jump into your programming sessions and be more productive while having more fun.