Tanks Artificial Intelligence

Portfolio

Author: Luuk Holleman  
Class: ISM5GPb  
Version: 4th of june 2015  
Course: AI for Games  
Teacher: Hieke Keuning

Contents

[Unity 3](#_Toc421224360)

[Game Mechanics 5](#_Toc421224361)

[Steering 6](#_Toc421224362)

[Seek/Flee/Arrive 6](#_Toc421224363)

[Wander 6](#_Toc421224364)

[ObstacleAvoidance 6](#_Toc421224365)

[Flocking, Separation/Alignment/Cohesion 6](#_Toc421224366)

[Pursuit 6](#_Toc421224367)

[Orbit 6](#_Toc421224368)

[Path Planning 8](#_Toc421224369)

[Behaviour/Fuzzy Logic 11](#_Toc421224370)

[FLV’s 11](#_Toc421224371)

[Attack 12](#_Toc421224372)

[Evaluator 12](#_Toc421224373)

[Goal 12](#_Toc421224374)

[CaptureFlag 12](#_Toc421224375)

[Evaluator 12](#_Toc421224376)

[Goal 12](#_Toc421224377)

[DefendFlag 12](#_Toc421224378)

[Evaluator 12](#_Toc421224379)

[Goal 12](#_Toc421224380)

[Flee 13](#_Toc421224381)

[Evaluator 13](#_Toc421224382)

[Goal 13](#_Toc421224383)

[GetPowerUp 13](#_Toc421224384)

[Evaluator 13](#_Toc421224385)

[Goal 13](#_Toc421224386)

[Conclusion 14](#_Toc421224387)

# Unity

I chose to do this project in Unity because I’ve wanted to learn it for 2 years now. Unity works a bit different than XNA, that’s why I’m writing a chapter about some basic info that you’ll need to understand the code.

In Unity, everything exists of GameObjects, the engine can only communicate with GameObjects, there is no main class where you place code. A GameObject has by default a transform with a position, rotation and scale. You can add behaviour to these GameObjects with Components. A Component can be for example:

* BoxCollider
* AudioSource
* SpriteRenderer
* LightSource
* Rigidbody – gives the GameObject physics.
* Etc.

These Components are scripts written in C#, JavaScript or Boo, but most programmers are using C#. We’re game programmers, not some peasant web developers! These scripts need to derive from MonoBehaviour and that’s all you need. When you’ve done this, you can attach this component to a GameObject in the editor.

In the component script you can make different methods which will get called by the engine. Some that I used are:

* Awake – called on creation of script
* Start – called on start of game
* Update – called every frame
* LateUpdate – called between update and frame render
* FixedUpdate – called on a constant base (default 60 hertz)
* OnCollisionEnter2D – called when the attached rigidbody component collided
* OnCollisionStay2D – called when the attached rigidbody component is still in collision
* OnCollisionExit2D – called when the attached rigidbody component is no longer colliding
* Every Collision method is also in the flavor of Trigger, a trigger is a rigidbody that does not get collision physics (they can move through each other)
* OnDestroy – called before the object is destroyed
* OnGUI – called when the GUI needs to be drawn, 2 times a frame

[http://docs.Unity3d.com/ScriptReference/MonoBehaviour.html](http://docs.unity3d.com/ScriptReference/MonoBehaviour.html) for more documentation about MonoBehaviours or <http://docs.unity3d.com/Manual/index.html> for the complete Unity documentation.

Because there is no way to check which GameObjects are using what Components without the Unity editor I made a list of the most noteworthy combinations:

* Tank
  + SpriteRenderer
  + RigidBody2D
  + BoxCollider2D
  + Tank
  + GoalComponent
  + ChatBubble
  + Barrel child GameObject
    - SpriteRenderer
    - Barrel
* Rocket
  + SpriteRenderer
  + RigidBody2D
  + Rocket
  + BoxCollider2D
* Flag
  + SpriteRenderer
  + Flag
  + CircleCollider2D
* Explosion
  + SpriteRenderer
  + Explosion
* PowerUps
  + SpriteRenderer
  + PowerUp
* Tree/Obstacles
  + SpriteRenderer
  + Circle/Box Collider2D

# Game Mechanics

In the world there are 5 flags to capture: North, East, South, West and Center. Tanks can capture a flag and score points for their team. The first team who reaches 500 points has won. If a tank dies, they respawn after 15 seconds at the nearest flag from where they die (if no enemy is capturing that flag), if they have no flags, the tank will spawn at the base.

Tanks capture a flag while they stand in a 5 unit radius of the flag (a tree is on average 5 units). A flags score is in the range of -10 to 10. -10 for red, 10 for blue. While the tanks are in the range of a flag, the score is skewed to the maximum score with a score of 0.1 per tank per second. Because the center flag is a strategic place, this flag is captured by a rate of 0.5 per tank per second. When a flag reaches the score of -8 or 8, the flag will transform in the respective color.

An example:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 5 red, 0 blue | 3 red, 10 blue | 5 red, 5 blue |
| Begin score | 0 | 0 | 0 |
| After 1 second | -0.5 | 0.7 | 0 |
| After 2 seconds | -1.0 | 1.4 | 0 |

The points awarded to a team are based on the amount of flags captured.

|  |  |
| --- | --- |
| Flags captured | Score per second |
| 0 | 0 |
| 1 | 0.8 |
| 2 | 1.1 |
| 3 | 1.7 |
| 4 | 3.3 |
| 5 | 30 |

If a tank defends a base, the damage done is increased by a factor of 1.3.

Power ups increase different aspects of the tank, currently there are 3 upgrades:

* Movement – increase movement speed by 1 (default 5)
* Reload – decreases reload time by 0.2 with a minimum of 0.1 second per reload
* Repair – fully repairs the tank to 100 health.

# Steering

After all I’ve had not much problems with the steering behaviours, it was just a copy from the book. Though it did took me a long time to realize Unity’s coordinate system is different than the one used in the book. In the book, up was the X coordinate and right was Y. In Unity it’s the other way around, X is right, Y is up. As you can imagine, this gave some really nasty and strange bugs.

## Seek/Flee/Arrive

No problems at all

## Wander

The difficult part about wander was actually the debugging while developing; it was the first steering behaviour that I developed that really had an algorithm. I didn’t know what all vectors represented so I did not know where to look for.

## ObstacleAvoidance

Hardest of all, I had no idea what all the vectors and calculations did and I didn’t found out yet about the turned coordinate system. I spent several days learning this behaviour to discover that I should switch the coordinates. Before that, the behaviour was pretty random, sometimes it didn’t care about objects at all, sometimes it did work and sometimes it moved towards the obstacle.

It does work now but it has not the power that I would’ve liked, sometimes it goes left while he needs to go right and vice versa. Maybe I could fix it but the tanks are using it too less to have a priority.

## Flocking, Separation/Alignment/Cohesion

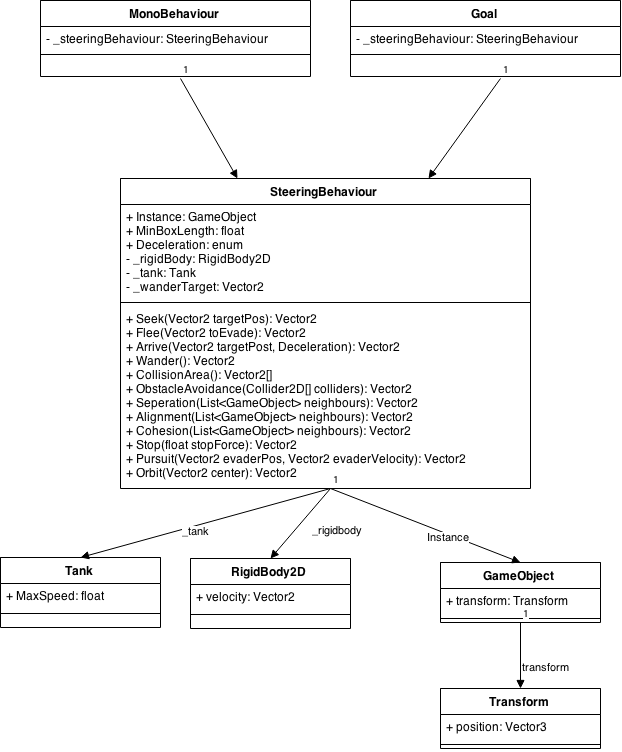
Easy, implemented in 1 try.

## Pursuit

I implemented this after several weeks so there were no problems at all.

## Orbit

Instead of a steering behaviour that explores the world (I don’t need that in my simulation) I made a behaviour that orbits around a given point. So it looks like the tanks are patrolling around the flag. What it basically does is it sets your velocity towards the center as the same speed you’re moving forward. If you take into account the current distance to the center, you’re moving in circles.

The class diagram of the steeringbehaviour

# Path Planning

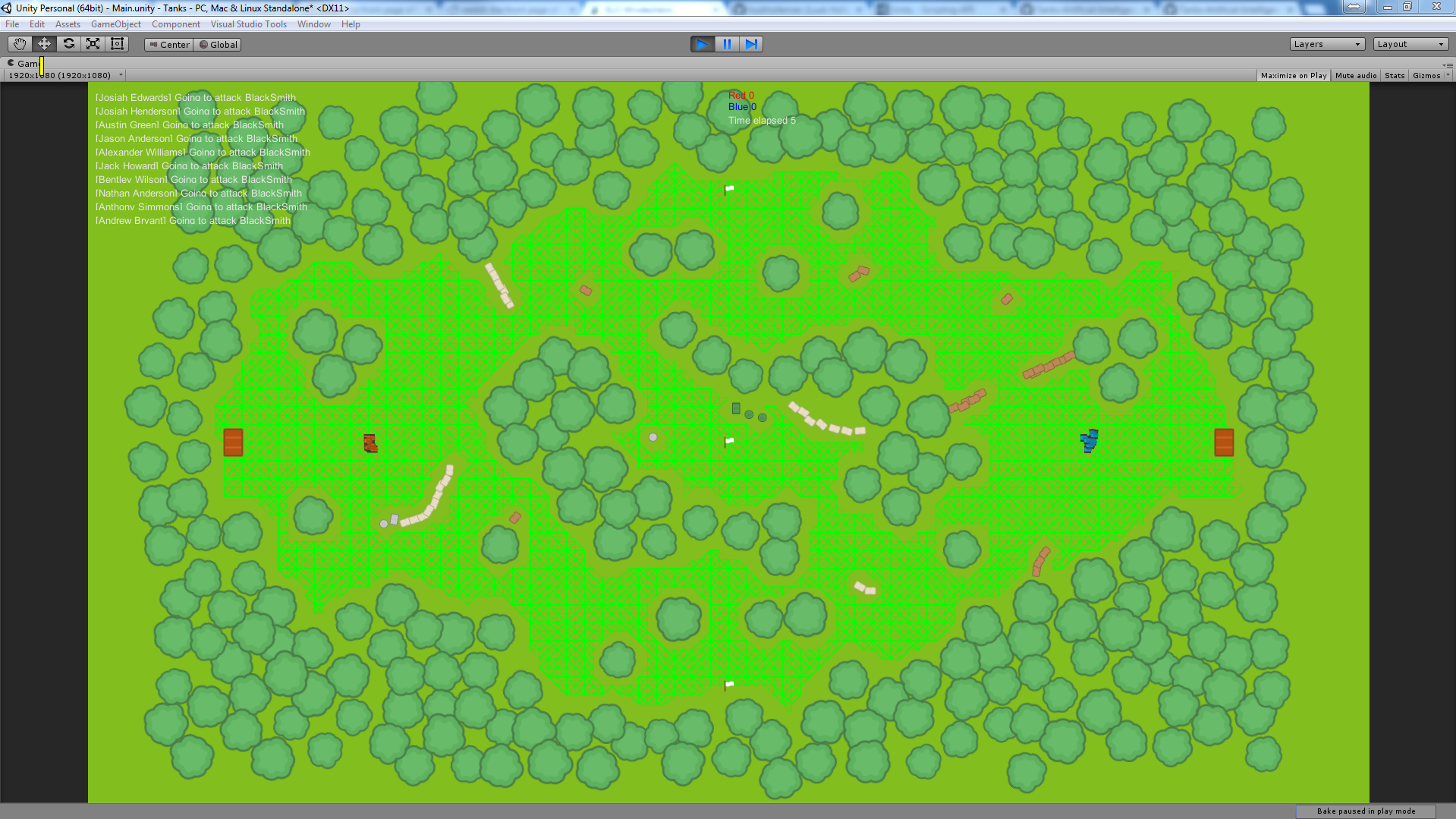
I used my own version of creating the graph. It basically is just a brute force, it checks every absolute coordinate if there is an obstacle, if not, add that coordinate to the graph and connect with the southwest, west, northwest and north node if present. We can skip the other orientations because we’re looping over the field and that nodes cannot exist yet.

I also made a floodfill because it was a requirement; I chose not to use it because it was way slower. It’s still present in the Graph component, just not executed. I ran tests with it, this were the results:

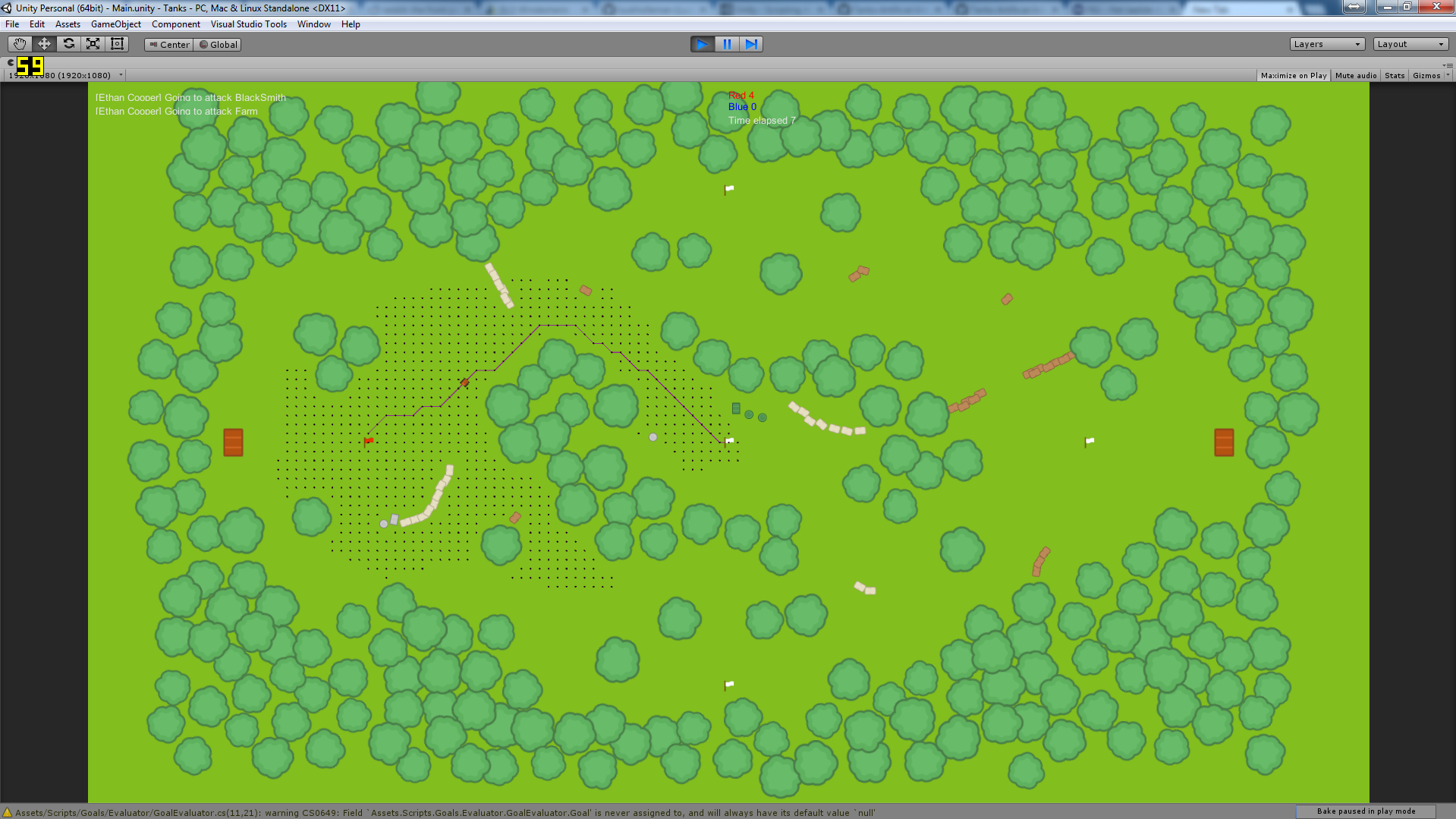
Brute force 3195 nodes in 2.6 seconds  
FloodFill 3223 nodes in 6.6 seconds

The floodfill does find more 28 nodes but that’s 0.89% of the total amount, that’s not worth the 4 second difference.

Created graph:



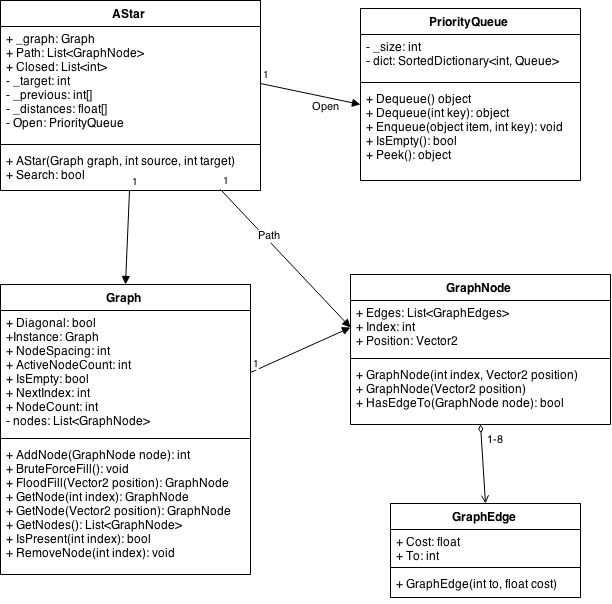
Path found from left flag to middle flag, the line is the path. Dots are the checked nodes.



You can activate the graph with G, the paths with P. Currently only the active paths are drawn; new paths only get drawn on another press. Be aware that pressing G or P is very slow; Unity can’t draw lines on production builds so everything has to be a quad.

One of the assignments was that you could order a tank to move to a specific point, this is not implemented as this is not possible without writing ugly code. I could give a tank the goal of going to a clicked point but that would immediately get overwritten by the evaluators.

Class diagram of the path finding:

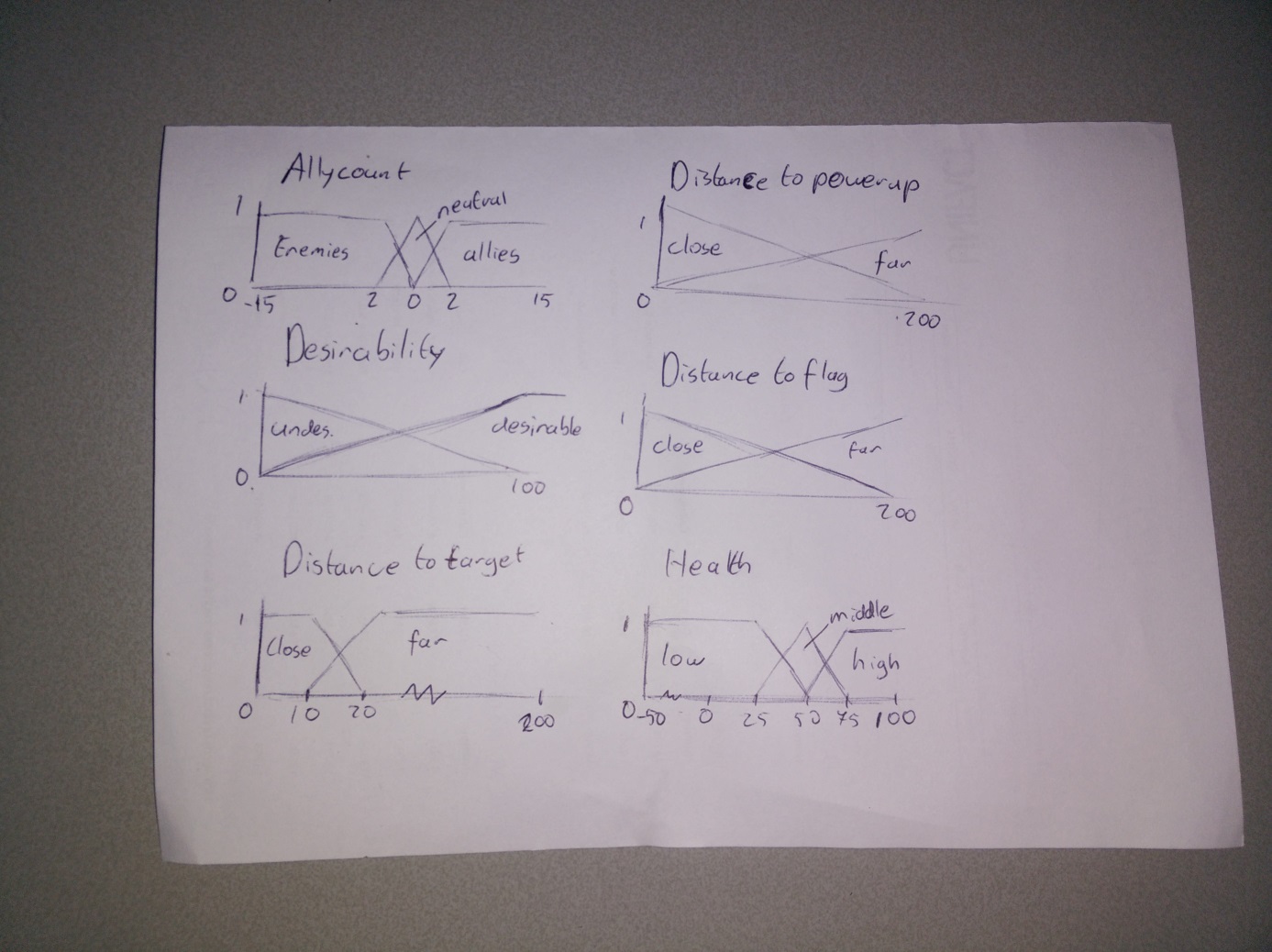


# Behaviour/Fuzzy Logic

I chose for option 1, goal driven behaviour. For this project I created different different goals which I will discuss separately. I wrote the fuzzy logic with Tim.

When you press D, you can see the chat message (current goal) of the tank.

## FLV’s



## Attack

Used for deciding to attack an enemy tank.

### Evaluator

The evaluator loops all tanks and skips allies. For each enemy tank the evaluator evaluates the desirability to attack it and returns the most desirable. It takes into account:

* Distance to target
* Current health
* Enemy health

### Goal

The attacking goal uses the pursuit combined with obstacle avoidance steering behaviours to get to the enemy tank. If it reached a distance of 2 units it stops the movement so the tanks do not stand on each other.

## CaptureFlag

Used for deciding to capture an enemy or neutral flag.

### Evaluator

Loops all flags and skips the flags that are already captured by his team. For each flag it calculates the desirability and returns the most desirable. It takes into account:

* Distance to flag
* Enemy/Ally ratio

### Goal

The capture flag goal uses two subgoals: FollowPath and WaitAtFlagTillCaptured. Follow path will of course; follow the path to the flag. When it reached the flag he will wait in the capturing range till the flag is captured.

## DefendFlag

Used for deciding to defend an ally flag.

### Evaluator

Loops all flags and skips the flags that are not captured by his team. For each flag it calculates the desirability and returns the most desirable. It takes into account:

* Distance to flag

### Goal

The defend flag goal also uses two subgoals: FollowPath and WaitAtFlagTillCaptured, The same as Captureflag. That is because the behaviour of WaitAtFlagTillCaptured also works on defending. If a tank goes to a flag that is captured by the enemy in meantime, it will immediately start to capture it back. If the flag still belongs to his team, the goal will succeed immediately and the Attack goal will trigger on surrounding enemies.

## Flee

Used for deciding to flee from a fight or incoming tank.

### Evaluator

The evaluator loops all tanks and skips allies. For each enemy tank the evaluator evaluates the desirability to flee from it and returns the most desirable. It takes into account:

* Distance to target
* Current health
* Enemy health

You would think flee is just the inverse of attack, and the first version actually was, but that is not the case. With an inversed evaluator there was a lot of fleeing and not so much fighting so tanks are more likely to attack and die than flee when it’s not needed.

### Goal

The fleeing goal uses the evade combined with obstacle avoidance steering behaviours to get away from the enemy tank.

## GetPowerUp

Used for deciding to attack a get a power up.

### Evaluator

The evaluator loops all power ups (if there are any). For each power up the evaluator evaluates the desirability to attack it and returns the most desirable. It takes into account:

* Distance to power up

### Goal

The attacking goal uses the FollowPath subgoal to get to the power up.

# Conclusion

I think it was a very fun project to do, if not the most fun of the entire study. I spend a lot of extra time on the project just because it was enjoyable. I’m sure you can also see this in the end result. I wanted to start with an empty field and some tanks patrolling that shoot at each other. But it grew to a battlefield with goals to capture and large clustered fights.

The book is also the first book that I’ve read for my own pleasure; it really is a very good book. All the subjects are very interesting and they’re explained very well.

Next year it would be really nice if you could group up with 2-3 persons so you can a really nice simulation.

There were of course also some down sides, some requirements are really strange. Like “If you click somewhere in the game world, your game character should be able to go there using the shortest path calculated by A\*”. I wonder why this is a requirement, it adds nothing of value to the simulation.

Also “Implement a new steering behaviour ‘Explore’. This behaviour shows a character exploring a certain area (your game world) in a systematic way”, these requirements force you to have a style of game.

What I’ve could improve on my simulation is that the tanks are dividing and conquering, although I have no idea how.