INTELLIGENCE ARTIFICIELLE POUR STARCRAFT - PROTOSS

INF584A

Rapport de projet

March 23, 2022

Michael PILCER, Josselin SOMERVILLE

ÉCOLE POLYTECHNIQUE

IP PARIS

1 INTRODUCTION

The bot aims for an "economic" victory. This means that it trains a lot of workers and rapidly creates a second base. However in the early game it is quite vulnerable (especially against a Zerg rush). To counter this, the bot tries to build Photon Canon quite rapidly.

After training a few zealots that are grouped near the second base, we rapidly move to the famous protoss army: dragoon/observers. We rush the singularity charge upgrade as dragoon are pretty much useless without it.

Then, we add a lot of upgrades for the dragoons and finally, after a lot of dragoons have been trained, we add in late game troups such as scouts and carriers. The army has several modes. With few units or under attack it gather its troups to defend, with a few more troups it send some units to harass the opponent and finally with a lot of troups it sends all units to attack the ennemy base.

All the transitions are handled by our "behaviour trees". There are dozens of thresholds to determine when to change the army composition. This is handled with requirements, the bot has to meet the requirements of the current composition and train enough troups to go to the next one.

When a building is destroyed, it is reconstructed only if it is necessary for the current army composition. Workers are always retrained upon destruction.

2
BUILD QUEUE

To handle all expenses, we created a build queue (handling building, training, upgrading and researching). the build queue is an ordered vector of build task. A Build task is represented as:

- something to build/train/research/upgrade
- a priority
- a location (position and if fixed or approximate)
- a boolean to tell if the task should be unique in the queue

A task goes then though the diffent steps:

- initialization
- requiring resources
- acquiring a worker *(only for building operations)*
- moving to position *(only for building operations)* start building
- waiting for the unit *(only for building operations)*
- building
- task complete

Several errors can occur during this process (the position is not valid anymore, the worker gets killed, ...), this is why it is possible to go back up one or several steps. You will find below a simplified behaviour tree of the update of a task (called every frame):

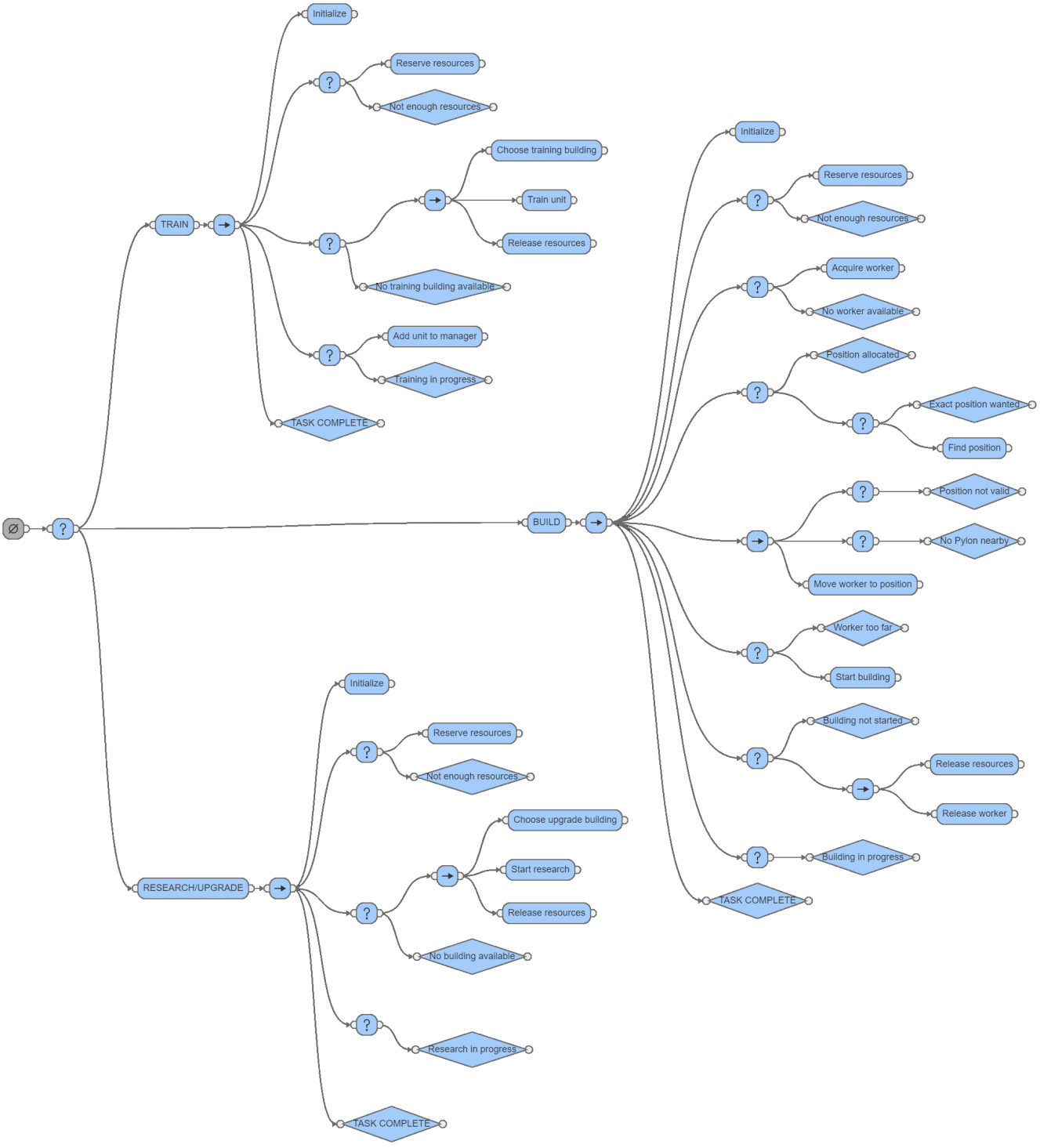

Figure 1: Behaviour Tree of a task

Handling the build queue is also a challenge in itself, especially adding tasks to the queue. The task has to be added in the rigt spot (with respect to its priority), no unique task should be added twice but instead updated and if some tasks cannot be performed (missing requirements), other tasks with higher priorities need to be added (such as asking for more supply, constructing a building to perform a research, ...).

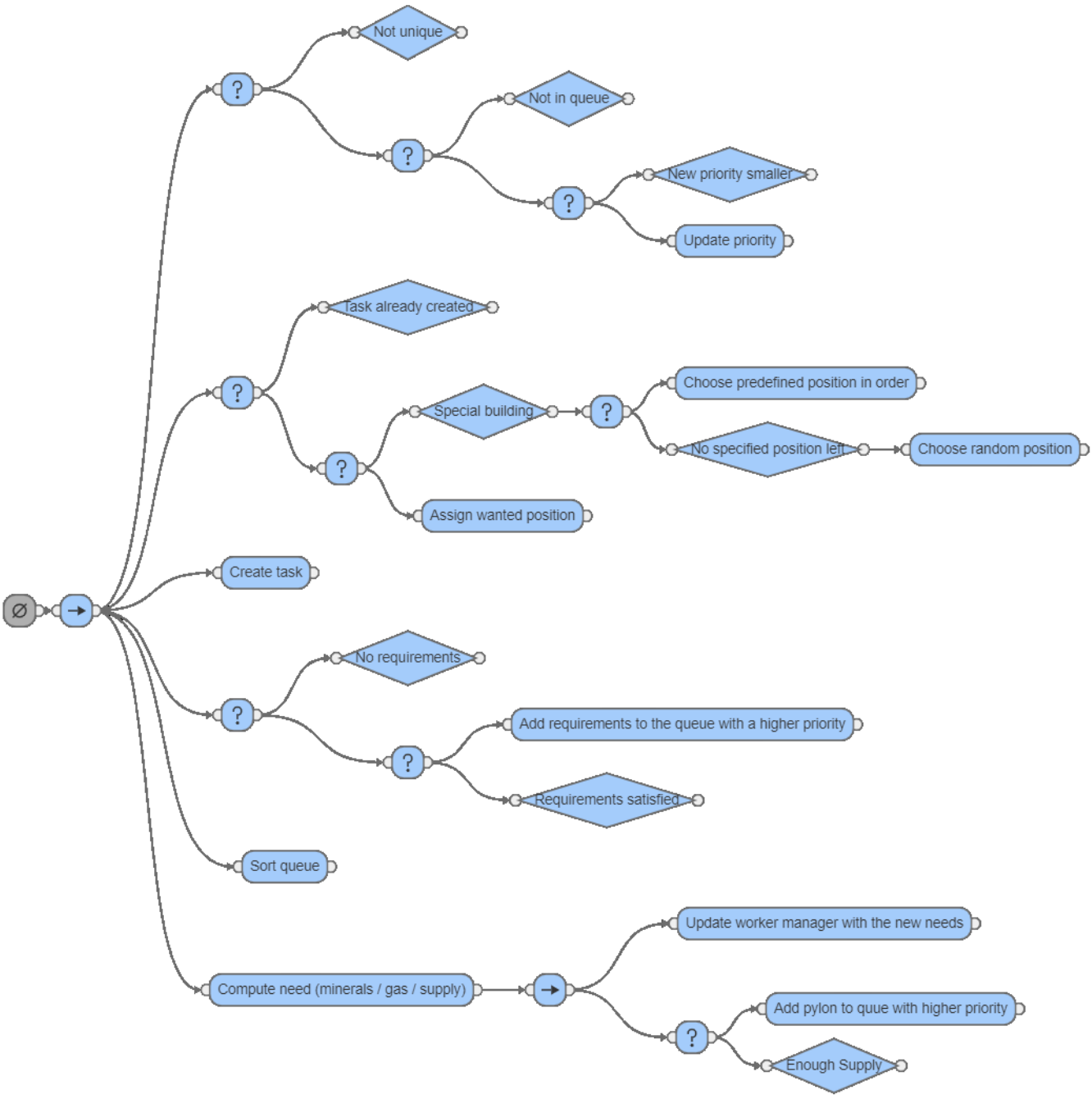

Figure 2: Behaviour Tree of adding a task to the queue

3
RESSOURCES MANAGEMENT

To handle resources, we chose to implement a **WorkerManager** handling all workers and the collection of resources. This manager is capable of assigning workers to minerals or gas, asking for more workers if needed, asking for the constuction of a refinery, asking for a second base and providing workers to construct buildings. Below, a simplified behaviour tree of the update function of the Worker Manager:

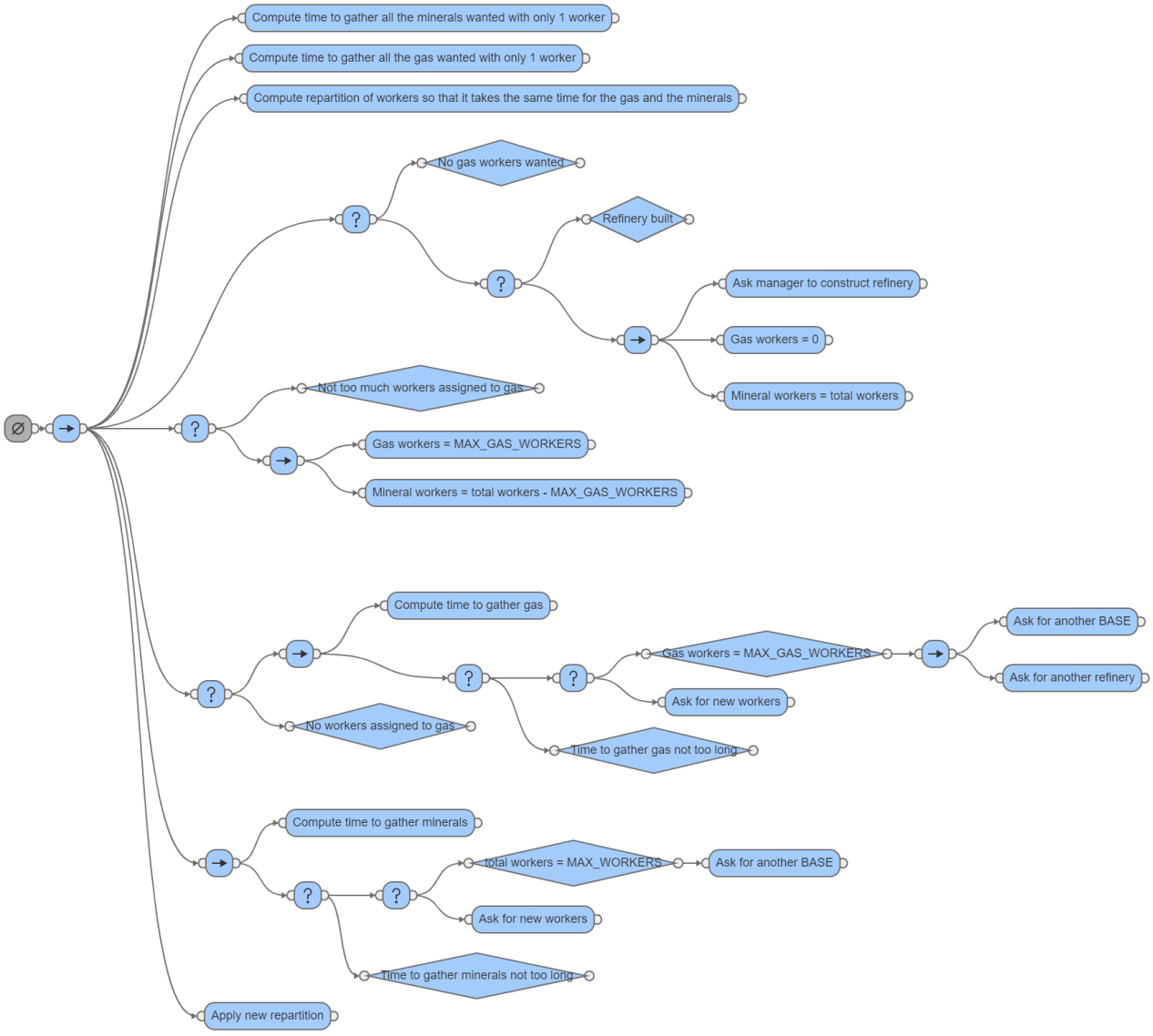

Figure 3: Behaviour Tree of the worker manager

4
ARMY MANAGEMENT

To handle all troups (i.e. all moving units except workers), we created an **ArmyManager**. This manager is responsible of asking for more units, choosing when to attack, when to harass and when to defend (thanks to its *"mode"*), which buildings to build for its army and what troups to train. Below, a simplified behaviour tree of the update of the Army manager:

Figure 4: Behaviour Tree of the army manager update function

5
GLOBAL COORDINATION

Finally, the bot needs to have a global coordination. This is done with the **GlobalManager** which takes car of updating all the bases, build queues and the army. The Global Manager is also responsible of ensuring the communication between bases, the worker manager and its base, and more...

Below is the simplified behaviour tree of the update function of the global manager *(many things occur outside the update function, with special events)*:

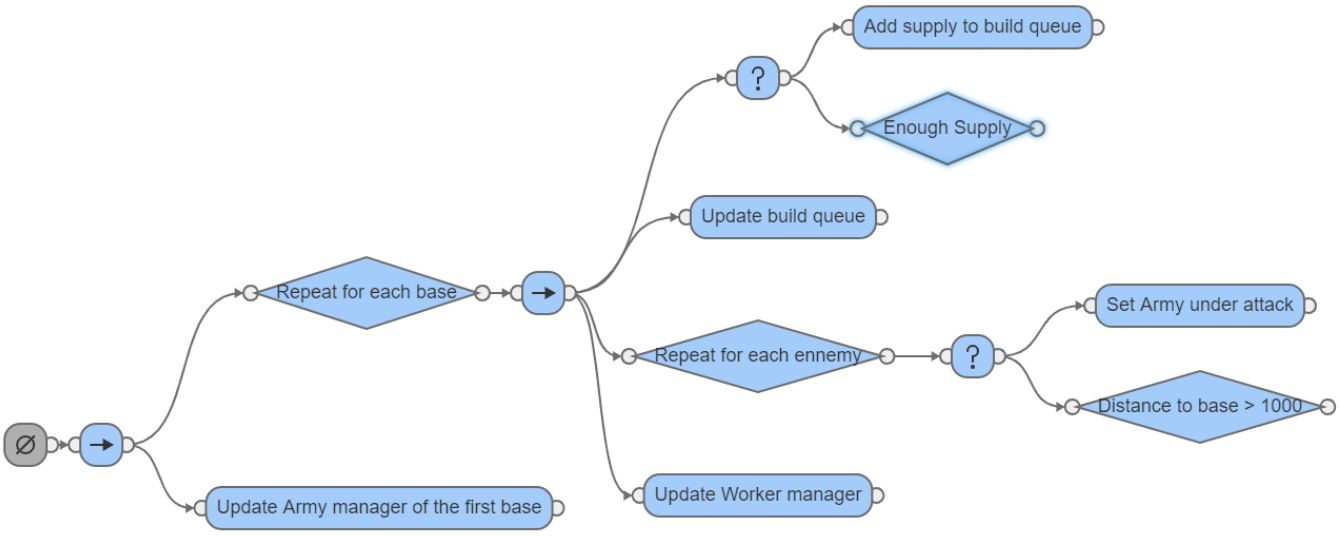

Figure 5: Behaviour Tree of the global manager