Advanced Programming Techniques
ExAdvPt
Winter Term 2016/2017

Project (part 1)

Introduction

In a typical match of the popular real-time strategy game Starcraft 2[1] up to eight human players compete in two teams and try to destroy the enemy base. A match can be divided up into three different phases: early, mid, and late game. In the first phase, the players try to build up their main base and decide for a certain opening strategy, where one can focus either on fast expansion to increase later income, developing straightly new technology to build better units, or quickly build as many units as possible in order to rush the enemy, i. e. to attack as soon as possible. The best opening strategy depends on the map, the race the participants have chosen, and also the opponents' strategies.

The goal of the AdvPt project will be to simulate and optimize one aspect of the early game, the production of buildings and units. The creation of buildings consumes time and resources, and possibly requires a certain technology, which is represented by the presence of other building types. Creation of units also occupies a building as production site for limited time, and generally requires supply, which is provided by some buildings.

The first milestone is to design a simulator that is able to catch the *important* events and effects *in the early game*. Therefore it is reasonable to simulate a simplified variant of the original game, and foremost neglect placement and movement on the map as well as enemy contact.

In the following we will briefly outline a minimal feature set for the simulator and introduce the three races of the StarCraft 2 universe.

It is reasonable to simulate with a constant time step (unit time), because the duration of all actions will be specified in full seconds. We assume that one can initiate at most one game action per second.

Resources

Three types of resources are common to all races: Mineral, Vespene gas, as well as supply. As floating-point types are not able to represent many common decimal fractions like 0.1 exactly, it is preferable to use fixed-point representations for the two former and plain integrals for the latter.

The start configuration always includes a few workers (SCV/Probe/Drone) and a basic building close to mineral patches and a Vespene geyser, but a special building (Refinery/Extractor/Assimilator) is necessary to harvest Vespene gas.

Use the following simple approach which is reasonable accurate for the early game and works without specifying positions and distances:

- Minerals are excavated at an average rate per harvesting worker ($\sim 0.7\ /\mathrm{s}$).

[1] `http://battle.net/sc2/`

- Vespene gas is similarly harvested according to average rates ($\sim 0.35\ /\mathrm{s}$), but not more than three workers must be assigned to this job simultaneously per tapped geyser. There are only two geysers available per base.

Rationale: In the real game, units are actually moving between the building and the resources, even occupying a patch or geyser for a short period. The actual crop that can be won from each depot therefore depends on the number of workers harvesting it and the distance to the initial building. As there are only two Vespene geysers, rate of yield is nearly maxed out with three harvesters at each. As there are multiple mineral patches, saturation effects occur later, and as we are interested in early game only, we can equally neglect that depots are actually finite.

Construction of buildings

All three races have different ways of constructing new buildings. Given that enough resources are available and all dependencies are met (Tech level), a worker is required for the construction:

Protoss: A probe must only move to the desired position to initiate the warp process, but can nearly instantly continue.

Zerg: After moving to the desired position, the Drone morphs into the respective building and is therefore effectively deleted.

Terran: SCV have to create buildings themselves and are therefore occupied for the whole construction time.

Neglecting time for movement altogether seems acceptable. Similarly, one can use a heuristic approach to assign workers, e. g. first subtract all currently occupied with building, using up to three for Vespene (if already unlocked), and the remaining for minerals. However you are free to implement a better strategy for worker assignment.

As combat situations and some advanced features are ignored, we need to consider the following features of buildings:

- For Terran and Protoss, units are created inside designated buildings. Usually this process is straight-forward and occupies the building for a certain period of time and consumes a certain amount of resources, and can additionally be dependent on a certain Tech level.
- Buildings can represent levels of technology and thereby enable the creation of more advanced buildings and units. Tech-only buildings serve no other purpose than that, and as merely all units and buildings of Zerg are morphed from another, this is the case for almost all Zerg buildings.
- Each race has an own building type that is necessary to harvest Vespene gas. As those are placed on a geyser and typical maps provide only two geysers per base, there cannot be more of those buildings than that. Workers can start harvesting only after such a building has been completed.
- Each race has its own collection of buildings or units that provide supply. The supply is only effective after the respective building has been completed. Most units require supply. Their supply cost is effective as soon as the build process for this unit starts.

- A number of buildings can be morphed or upgraded into a new building, especially Zerg and some Terran. This transformation can only start when the building is idle, and the building must not be used until the transformation time has passed.

Specialties

Each race has a different way of creating units and distinct abilities.

Protoss units are created in buildings straight-forward. If the necessary resources are available and dependencies are met, an idle building with the respective capability can initiate the process. After the production time, the unit becomes available and the production site is usable again.

Every Nexus building loads up on energy with a rate of $0.5625\ /\mathrm{s}$ (this rate is common to all races, units and buildings) to a maximum of 100, starting out with no energy. At the cost of 25 energy, a Chrono Boost can be executed, which speeds up the production rate of one arbitrary building by 50 $\%$ for $20\ \mathrm{s}$.

We do not model the Warpgate upgrade of the Gateway, because it gets more efficient when more powerful units are to be created when the game progresses. Modeling its warp-in and cool-down behavior is quite cumbersome.

Terran units are created similar to Protoss, but besides increasing capabilities with Tech levels, Terran Barracks, Factories and Starports can be extended through add-ons. The Reactor allows to produce two units at a time, while the Tech Lab enables the production of more advanced units. In contrast to Tech level upgrades, add-ons affect only a single building which must be idle for the time of building the add-on. In that sense, the add-on is not built by an SCV worker, but by the respective building. A building can have no more than one add-on.

Note: Most Terran buildings can lift off and thereby, leaving their add-ons behind, effectively swap them. We do not model such actions, we instead treat the combination of building and add-on as a virtual building type.

Similar to Nexus, an Orbital Command (an upgrade of the Command Center) loads up on energy, starting out with 50 up to a limit of 200. The mandatory feature is the *Calldown: MULE* consuming 50 energy. A Mobile Utility Lunar Excavator can be used for repair (not of interest to us) or for harvesting minerals, but is available only for $90\ \mathrm{s}$. MULEs require no other resources or supply and do not conflict with harvesting SCVs, but provide the yield of roughly four SCVs.

Optionally you can provide the equally expensive *Calldown: Extra Supplies*, which takes $4\ \mathrm{s}$ to become active and gives a permanent, one-time increase to a Supply Depot of eight supply. For each supply depot this can be applied only once.

Zerg differs not only in the creation of buildings, but also for units. Larvae are created by various mechanisms, and can —given the necessary resources— be morphed into basic units and further into advanced units and buildings etc.

"Naturally" larvae spawn from Hatcheries (and their upgraded variants Lair and Hive) every $15\ \mathrm{s}$ to a maximum of three at the respective building. The second important mechanism is the ability of the Queen unit, which is the only unit explicitly created in Hatcheries like with the other races, and that

unit starts out with 50 energy and has maximum of 200. For us, its *Inject Larvae* action is important, as it allows the Queen at the cost of 25 energy to place four Larva eggs into a hatchery, which will become Larvae after $40\ \mathrm{s}$. This way, a hatchery can pile up on more Larvae, but not more than 19.

Remarks: There is a number of effects that are deliberately ignored, besides others SC2 players will notice the absence of Creep, dependence of Protoss buildings on Pylons, the Nydus worm, and much more. Additionally all combat-related actions and upgrades (armor, impact, speed, special attacks, . . .) have been left out.

Task 1: Class design

Design classes for a SC2 forward simulator, preferably as UML diagram(s). In principle you can use a single omnipotent class or write a class for every single object type down to Larvae. Probably better designs are somewhere in between, defining a limited set of classes with diverse functionalities, which are optimally configured dynamically through configuration files.

The actual control logic will be added later, but your design shall allow to step through time, check amount of resources and objects, and if an action can be executed (building, morphing, special action) and actually issue it.

You do not have to submit your solution to this task, however you should show your design diagrams to your AdvPt tutors during the computer exercises. Use this opportunity to get early feedback on your software design. Changing your design at a later stage in the project means lots of work.