Dr. Harald Köstler
Markus Stürmer

Winter Term
2012/2013

Advanced Programming Techniques

Project Assignment

Optimizing build orders in a real-time strategy game

Electronic sports, i. e. competitive play of video games, attracts more and more people, and Starcraft 2[1] is one of the most popular real-time strategy games currently. In a typical match, 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.

In this project we focus on a certain 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, represented by the presence of other building types. Creation of units also occupies a building as production site for limited time, and generally requires support, which is provided by certain buildings.

Neglecting the actual placement and movement of the items on the map, the task in this project is to find the best build order for buildings and units for a slightly simplified rule-set for the *Protoss* race. A solution is then a build list like the one shown in figure 1. The necessary Techtree, i. e. which dependencies exist for units and between buildings, is shown in figure 2.

Figure 1: Possible build list. Units/buildings are produced one after each other from left to right.

The project is divided into three parts, where the first and the second part are obligatory for all groups, the third part is optional.

1. On our website we provide an example input file that describes all available building and unit properties in detail. This includes
   - the name of the building or unit,
   - the amount of Psi (support) it provides (non-zero only for buildings),

[1] `http://eu.battle.net/sc2/en/game/`

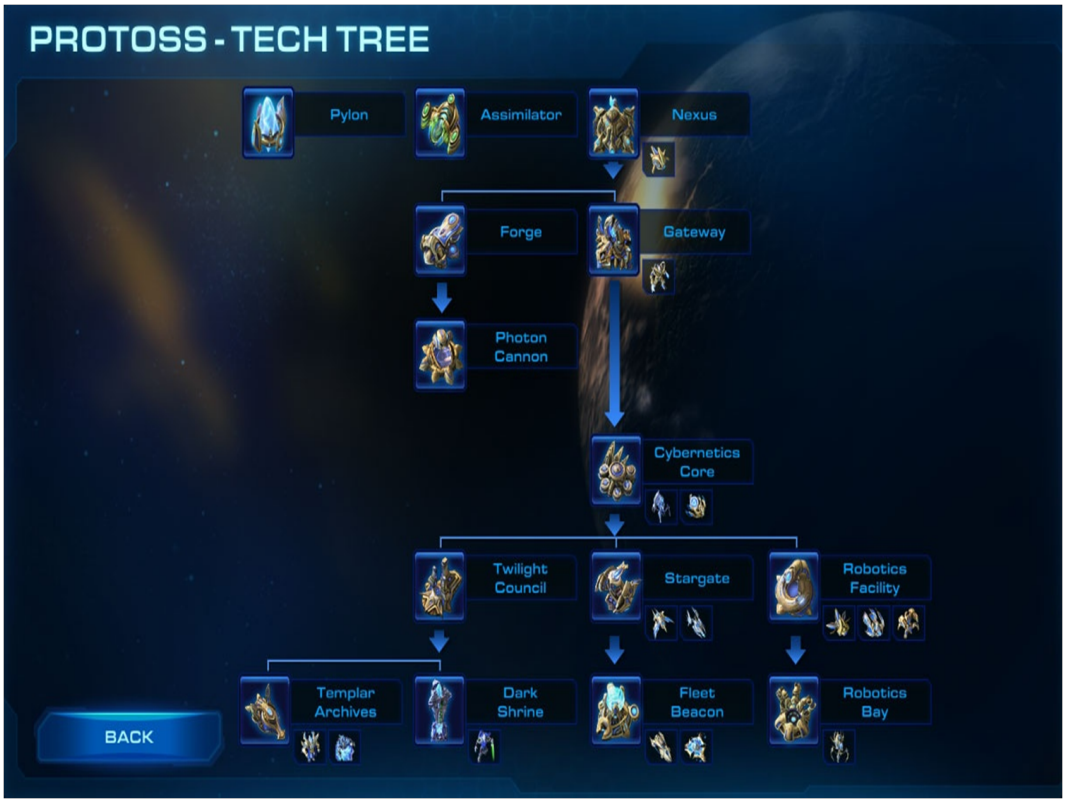

Figure 2: Full Techtree for the Protoss race in Starcraft 2. It shows which buildings are prerequisites to others and which units can be built in a certain building. In the project we start with a subset of this Techtree.

- the amount of Psi it occupies (non-zero only for units – if there is currently not enough Psi available, the unit cannot be built),
- which resources (minerals and Vespin gas) are consumed by the production of the object,
- how long the production takes (in seconds $\approx$ time steps, see below),
- which building the unit is produced in (dash means no building is occupied, this is generally the case for buildings),
- and which buildings (i. e. technologies) are prerequisites.

Read the file using standard library I/O and store the information in suitable data structures.

2. Implement a forward simulation that computes the exact build times for the given build lists found on our website (e. g., Probe Probe Pylon Assimilator Gateway Zealot) in the following way:

   - a discrete time stepping should be used, where each time step is one second,
   - one starts with one Nexus building and six Probe units (the Nexus provides 10 Psi, 6 of which are already occupied by the Probes),
   - initially an amount of 50 minerals and 0 Vespin gas are available,

- in each time step, one has to check if the production of one or more objects has been finished (do not forget the additional Psi provided by new buildings),
- every second / in each time step the production of at most one object can be started,
- every second resources are harvested, 1 unit of mineral is added per Probe and 3 units of Vespin gas per Assimilator.

The simulation terminates if production of the next entry in the given build list is impossible or after the last entry in the build list has been built. The output should be in which time step each item in the build list was completed (e. g., Probe 10s Probe 20s Pylon 100s Assimilator 130s Gateway 170s Zealot 200s).

3. Now the user provides a certain (maximal) time and unit type(s) he wants to produce in large quantities. The goal is to find the corresponding optimal build list. As an example the question could be the creation of a many Zealots as possible in 300 seconds. The simplest approach is just to test all possible build lists and select the one with the most Zealots. Since the number of all build list that can complete in a certain time explodes very quickly, this is not possible. Another approach would be to generate random build lists and select the one with the most Zealots. Here, there is a big chance not to find the optimal number of Zealots.

   In the lecture we will present a genetic algorithm to find the optimal build list. Implement this algorithm and test it for the given questions found on the website.