Goal-Reasoning in StarCraft: Brood War through Multilevel Planning

Moisés Martínez
King's College London
London, United Kingdom
moises.martinez@kcl.ac.uk

Nerea Luis
Universidad Carlos III de Madrid
Leganes, Spain
nluis@inf.uc3m.es

***Abstract*—Real-Time Strategy video-games (RTSVGs) are challenging for most deliberative approaches, such as Automated Planning. This is due to (i) the dynamic changes of the environment; (ii) and the wide variety of potential actions that can be performed over the environment. The aim is "*to win the match*". Besides, RTSVGs presents an additional challenge: managing goals during the game is extremely hard. They change as the game state evolves either because of actions performed by the different agents (player and opponents), by new available information or by unexpected changes of the environment. Thus, generating a detailed sequence of actions –plan– to win the match is not effective in the long term.**

In this paper, we propose an autonomous approach based on two levels of declarative Automated Planning. They are included inside a planning and execution architecture. The high-level, macromanagement, searches and suggests a set of soft-goals according to the current state and the available features of the agent. The low level, micromanagement, generates short plans of actions to reach the soft-goals generated by the high level. We claim that the ability of self-generating goals improves the plan generation and execution performance in a dynamic environment. Finally, we present a preliminary empirical evaluation of this approach tested on StarCraft: Brood War.

***Index Terms*—Automated Planning, Goal-Reasoning, Planning and Execution, Cognitive Systems, Video-Games**

I. Introduction

In recent years, video-games have gotten attraction by the Artificial Intelligence (AI) research community, as they offer challenging environments to test different AI techniques. Specially, StarCraft, which has become popular due to competitions such as SCAAIT [1], AIIDE[2] and the CIG Competition[3]. These competitions provide APIs and tools to easily deploy different AI approaches into StarCraft.

RTSVGs are considered a sub-genre of strategy games e.g. chess, backgammon, go, monopoly, etc. in which players develop a settlement based into two principles: (i) an economy i.e. exploitation of natural resources to build throughout the environment; (ii) and a military power i.e training different units and researching technologies which offer advantages over the enemy. These features hinder the application of traditional AI approaches i.e. the game complexity is extremely large in terms of the size of the state space and the search space and in the number of actions that can be executed on each decision step. For instance, a StarCraft map is defined as a rectangular grid where each tile is measured in building tiles. The smallest map has 64 x 64 building tiles. Thus, if a player builds the barracks (3x4), there exist 4096 tiles where they can be built in the worst case. If the player has four workers, there are 16384 instantiated actions available. Besides, a StarCraft match is executed at 24 frames per second, which means that the game state changes every 42 ms. The environment is non fully-observable, players just know the part of the environment that has been previously explored or built. In addition, the StarCraft environment is continuous, which means that players' turns do not explicitly exist. Actions are executed when requested by the players. Some actions can be executed in parallel. Besides, actions in this kind of games are durative. Thus, there is a small delay after the player chooses the action until it is executed on the game.

In order to win a match in a RTSVG like StarCraft, we must generate and execute a sequence of repetitive actions (plan) to defeat the opponent. But, to achieve this final goal, we must achieve first a huge number of small goals which appear during the game based on our previous actions and the new information discovered. Automated Planning (AP) is able to generate a sequence of actions that solves a specific problem like this one. Goals are usually given as input or as a part of the problem representation. However, in complex environments such as StarCraft, it is not possible to initially define a set of goals due to two important reasons: (i) the dynamism of the environment; (ii) the capabilities of the player, improved or penalized according to the actions executed over the environment. Nonetheless, several approaches based on AP have been successfully used before to solve complex problems with partial information such as the planning Mars exploration missions [1], managing fire extinctions [5], controlling underwater vehicles [20], intermodal transportation problems [8] and controlling quadcopters [3]. The key was to include some specific knowledge that simplified the deliberative act of reasoning.

In this paper, we propose a two level planning architecture. The high level consists on a goal-reasoning process, which generates goals based on the available information from the match e.g. player, environment, opponent etc. The low level consists on a deliberative process, which generates small plans

[1]http://sscaitournament.com/
[2]http://www.cs.mun.ca/ dchurchill/starcraftaicomp
[3]https://sites.google.com/site/starcraftaic/

that should reach the goals suggested by the higher level. These two levels are included into a goal-reasoning, planning and execution loop, which helps to control and process the real situation of the game to quickly update the plans.

This paper is organized as follows: first in Section II, we describe the planning framework. Then, Section III presents a description of the application domain. Section IV describes our contribution: the goal-reasoning process to control a StarCraft player. Later on Section V, we show preliminary experiments of our approach in different maps. Section VI contains the related work. Finally, in Section VII we present some conclusions and future work directions.

II. Planning Framework

In this section, we describe the different planning formalisms that define the Multilevel Planning approach. It considers two different kind of planning tasks: (i) a sequential classical planning task, which is encoded in the propositional fragment of the standard Planning Domain Description Language (PDDL) 2.2 [6]; and (ii) a temporal planning task, which is encoded in the propositional fragment of the standard in PDDL 2.1 [7]. In PDDL, a planning task is described in terms of objects of the world (units, buildings, mineral, etc), predicates which describe static or dynamic features of these objects (e.g. building are locations in a specific position), actions (a unit can move from one location to another, an unit can be training in a building), an initial state that describes the initial situation, and a goal definition which describes the objectives that must be reached (goals).

Definition 1: **(Planning task)**. A planning task can be defined as a tuple $\Pi = (F, A, I, G)$, where:

- F is a finite set of grounded literals (also known as facts or atoms).
- A is a finite set of grounded actions derived from the action schemes of the domain.
- $I \subseteq F$ is a finite set of grounded predicates that are true in the initial state.
- $G \subseteq F$ is a finite set of goals.

Any state s is a subset of facts that are true at a given time step. Each action $a_i \in A$ can be defined as a tuple $a_i = (Pre, Add, Del)$, where $Pre(a_i) \subseteq F$ are the preconditions of the action, $Add(a_i) \subseteq F$ are its add effects, and $Del(a_i) \subseteq F$ are the delete effects. $\mathit{Eff}(a) = Add(a) \cup Del(a)$ are the effects of the action. Actions can also have a cost, $c(a)$ (the default cost is one). An action a is applicable in s_i, if $Pre(a) \subseteq s_i$. The result of applying a in a state s_i generates a new state that can be defined as: $s_{i+1} = (s_i \cup Add(a) \setminus Del(a))$. A plan π is an ordered set of actions (commonly, a sequence) $\pi = (a_1, \ldots, a_n), \forall a_i \in A$, that transforms the initial state I into a state s_n where $G \subseteq s_n$. This plan π is valid if the preconditions of each action are satisfied in the state where the corresponding action is applied; i.e. $\forall a_i \in \pi, Pre(a_i) \subseteq s_{i-1}$ such that from applying the action a_i in the state s_{i-1} the system transits to s_i. The state s_0 represents I.

Definition 2: **(Temporal Planning task)**. A temporal planning task is a tuple $\Pi = (F, A^d, I, G)$ where

- F is a finite set of grounded literals (also known as facts or atoms) and numerical functions.
- A^d is a finite set of grounded durative actions.
- $I \subseteq F$ is a finite set of grounded predicates that are true in the initial state and a set of numerical functions.
- $G \subseteq F$ is a finite set of goals.

Each action $a_i \in A^d$ can be defined as a tuple $a_i = (D, Pre, Add, Del)$, where $D(a_i)$ is the duration of the action a. $Pre(a_i) = \langle Pre_s(a_i), Pre_o(a_i), Pre_e(a_i)\rangle$ where $Pre_s(a_i)$ are preconditions given at the start of a; $Pre_o(a_i)$ are preconditions given during the action's duration; and $Pre_e(a_i)$ represents preconditions given at the end of a. $Add(a_i) = \langle Add_s(a_i), Add_e(a_i)\rangle$, where $Add_s(a_i)$ are the add effects at the start of a_i and $Add_e(a_i)$, the ones at the end. $Del(a_i) = \langle Del_s(a_i), Del_e(a_i)\rangle$, where $Del_s(a_i)$ are the delete effects at the start of a_i and $Del_e(a_i)$, the ones at the end.

III. Description of the real time environment

Our approach has been applied to StarCraft: Brood War [4], which is a popular RTSVG that runs a science-fiction universe where three races (*Terran, Protoss* and *Zerg*) are at war. The aim is to defeat all your opponents. Players start the match in a random position of the map. During the match, the player exploits the resources in order to (i) train new workers; (ii) build new buildings - which allow players to train soldier units, unlock stronger units and unlock new buildings-; and (iii) research new technologies - improve units' features or unlock new skills-. Besides, the player has to explore the environment to discover new resources, expand its territory and find the opponents' position. In the end, players must define a global strategy in order to destroy every opponent's units and buildings.

Our definition of global strategy is composed by simple actions. First, actions related with units' control (e.g. collect, move, train, build, explore, research and attack) for micromanagement. Second, actions related with the global strategy (e.g. which kind of unit must be trained, which kind of building must be built, when and where to attack) for macromanagement. For instance, in order to build a tank we need first to build a *Machine shop*, a *Factory* and a *Command Center*. Thus, we need at least one worker to build the four buildings on four suitable locations in the map -that need to be explored first-. For instance, when the action "*train a tank*" is executed, a set of micromanagement actions were performed first. From the planning perspective, the macromanagement actions can be considered as dynamic soft-goals, which are generated during the match according to the information provided by the environment and the player; and the micromanagement actions are the sequence of actions of the plan π that reach the soft-goals. that

IV. Goal-reasoning through Multilevel Planning

StarCraft is a RTSVG where two or more players must compete for the resources of the environment in order to win

[4] https://starcraft.com

the match. From the point of view of planning, a StarCraft player should execute a sequence of actions – plan – provided by the planner to defeat all its opponents. However, generating a sequence of actions in a dynamic environment with partial information arises some issues: (i) new information about the environment is usually discovered during the action's execution; (ii) actions must be quickly chosen to interact within the environment as fast as possible - the sooner it happens, the greater the similarity with the current planning state; and (iii) goals should be generated dynamically during the game because of the lack of information. As we described before, the main goal of StarCraft is to win the match. Thus, the first issue we faced is how to model the PDDL domain to reach this hard goal. We are certain that this could not be satisfied if we just used classical planners. As the goal *win the match* is an abstract goal, it cannot be reached from a specific action executed by player. Neither a group of actions executed sequentially or in parallel guarantees to win the math. As the environment changes fast, it makes useless any long-term plan.

Our approach solves this issue by having a process that generates soft goals. They could be easily reached (or might not) according to the state of the match. Also, the agent has "high-level" tasks such as move, build, train or attack. Those can work as soft-goals as long as there is some low-level process to handle them on the game.

In previous works, the goal-reasoning problem has been addressed from different directions: (i) using an independent goal-reasoning system which receives a model of the environment, the current state and at least an initial goal, and returns either the same goal or a new one [14]; (ii) learning a model by means of Machine Learning to predict which goals must be achieved in order to control a real-time system [18]; and (iii) performing a hard goal life-cycle composed of different steps (selection, expansion, commit, dispatch, and finally execution) [10], [21]. In this paper, we propose an autonomous approach based on multiple levels of planning [17] that encapsulates the complexity of the domain in different layers.

The multilevel planning system is composed by two levels (macro and micro) of planning. The macromanagement level (high level) employs an abstract representation of the environment (numerical and logical) in order to define the set of soft-goals. This level uses temporal planning, which provides a richer version of PDDL e.g durative-actions, numerical fluents in preconditions and preconditions, etc. The output plan is a parallel plan, which explicitly indicates when two actions can be executed in parallel. Additionally, the soft-goals generation allow to use different policies for the generation of the plan in the micromanagement level. Figure 1 shows an example of an action defined in temporal PDDL which generates a goal `build-building(?b, ?l)` where `b` is the type of the building and `l` is the technological level needed to build it. As a result, this action generates goals to build any kind of building except for *Refinery* and *Supply Depot* which have special properties.

The temporal planner needs three elements to work: (i) an initial state; (ii) a specific domain that accurately describes a set of actions that, when instantiated, will serve as goals to "guide" the low-level planner; and (iii) a set of goals in order to stop the search process when they are satisfied. We were inspired by the limited horizon search approaches [11], [13] to define the stop criteria, which in our case is the upper-bound of goals that can be generated. We defined a fluent called *(max-number-goals)* as a counter that decreases in one each time an action-goal is generated. It stops in zero. The sequence of actions generated by the temporal planner is transformed into goals, which are used by the low planning level. Each action of the plan is considered like a predicate which must be reached in the next level.

```
(:durative-action generate-build-building
 :parameters (?b - building
              ?l - level
              ?a - area)
 :duration (= ?duration
              (* (building_cost ?b)
              (number_buildings ?b ?a)))
 :condition (and
    (at start (> (free_units scv) 0))
    (over all (> (max_number_goals) 0))
    (over all (tech_level ?l))
    (over all (tech_level_building ?b ?l))
    (over all (can_build ?b)))
 :effect (and
    (at end (decrease (max_number_goals) 1))
    (at start (decrease (free_units scv) 1))
    (at end (increase (free_units scv) 1))
                      (at end (have_building ?b ?a))
    (at end (increase (number_buildings ?b ?a) 1))
    (at end (increase (goals_score)
                      (score_building ?b)))))
```

Fig. 1. Example of a durative action that generates building goals.

The micromanagement level (low level) uses a structured representation of the environment to reach the previous soft-goals. This level employs classical planning in order to reach the goals. The plan described in Figure 3 is transformed into a group of three goals: (1) train a worker *(SCV)* in a *Command Center*, (2) build *Barracks* and (3) build a *Supply Depot*. Each goal needs a sequence of detail actions to be reached. The training goal needs two actions: one to chose a building of type command center (if there are more than one) and start training process of a *SCV* unit. After that, each building goal needs four actions to be reached: one action to choose an available worker that will perform the building process, a second action that chooses an available location using a external search algorithm, a third action that moves the worker to the building location and finally a fourth action which starts the building process. The agent will interleave goal and plan generation in a infinite loop using the PELEA architecture [5], which now implements the macro and micromanagement levels into its different modules. Therefore, in order to handle more complex situations we are using different planners working at

[5]The architecture described here is an instantiation of the original version [19] which employs two different representations of the environment in PDDL.

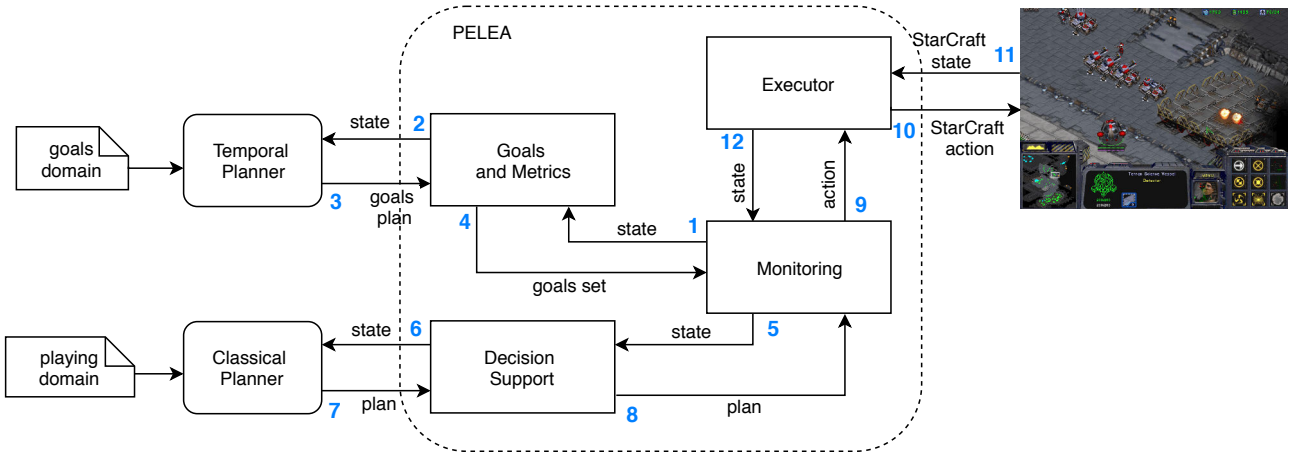

Fig. 2. Overview of the PELEA architecture applied to StarCraft. The multilevel planning approach is shown on the left of the Figure, where the temporal planner is in charge of generating the goal-based actions' plan (macromanagement level) and the classical planner generates the sequence of game actions (micromanagement level). Numbers in blue indicate the flow of the architecture from the initial state of the game. First, *Monitoring* asks to the *Goals and Metrics* for the high-level plan (initially the set of goals is empty); then the plan is transformed into a set of goals and is sent to the *Decision Support*, which generates a low-level plan based on those goals. The resulting plan is sent back to the *Monitoring*. This module sends the plan to the *Executor* action by action. When each action is executed, the game-state is updated and processed.

```
0.000: (train-unit scv command_center tl_one)  [7.0]
7.001: (build-building barracks tl_one)  [22.0]
29.002: (build-building supply_depot tl_one)  [8.0]
```

Fig. 3. Example of a high-level plan, whose actions will be used as the goal set for the low level.

different levels of abstraction. This version integrates planning, execution, monitoring and goal generation. Figure 2 shows the structure of the agent which is composed of four sub-modules.

- **Monitoring**: it is the central module of the planning agent. It synchronizes the communications among the other modules and monitors the action's execution i.e. dispatches the next action to *Execution* Module, requests for a new plan to the *Decision Support* and checks for differences between the expected state and the observed state of the environment sent by *Monitoring*. If an observed state is not valid, this module has to start another planning episode to generate a new plan according to the observed state.
- **Executor**: this module runs the actions into the environment by means of the *Brood War Application Programming Interface* (BWAPI)[6]. BWAPI provides a Java Native Interface to the Brood War API that uses the shared memory bridge to give our code access to the game state. Information that can be easily extracted through the API such as units' state, resources location; or some orders can be issued through training or moving commands. On every frame, JNI-BWAPI sends a game state update to the Java AI agent and waits for a response, which will contain a set of commands to execute. Besides, this module is responsible for initiating the goal-reasoning, planing and execution loop by sending to the *Monitoring* a particular problem and domain definition to be solved. Both the problem and the domain are described in PDDL.
- **Decision Support**: this module generates a plan of actions by the invocation of a classical planner (Metric FF [9]). When the *Monitoring* informs about a discrepancy between the observed state and the expected planning state, the *Decision Support* invokes the classical planner to generate a new plan.
- **Goals & Metrics**: this module searches and suggests different reachable goals according to the available information about the environment by using a temporal planner (OPTIC [2]). Additionally, this module keeps a registry of the different goals that have been solved in order to generate new goals when all of them have been reached.

A. Planning, Monitoring and Execution Architecture

We have developed a Planning and Execution agent based on PELEA. We contribute with an extended version of the *Goals and metrics* module

[6]https://code.google.com/p/jnibwapi/

B. Modeling StarCraft with PDDL

StarCraft uses a lot of complex (numerical and logical) information about the players and the environment to describe the current state of the game. We have described the environment using two abstraction levels that are modelled in PDDL. Our representation of the game state is modeled by using a sequence of binary and numerical values which represent the number of units, buildings and resources available for the players (our player and the opponents) located in the different areas of the map. The specific location of the units or their life points is not relevant for the planning process. Thus, they are omitted.

The game state is described using a sequence of variables (numerical and binary) which represent the units of each player. The map environment is divided into areas of same size. Commonly each area has a set of resources which can

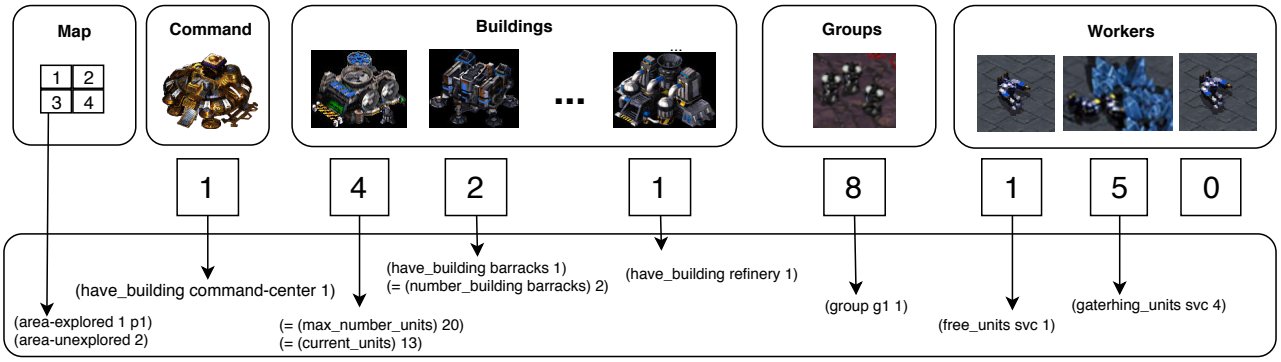

Fig. 4. Transformation of the partial state of the world from StarCraft to PDDL. The numerical sequence of values represents the current state of the game. Positions on the sequence correspond to one of the categories on the top of the Figure. Numbers are mapped into PDDL predicates as in the bottom part of the Figure.

be exploited. The units and buildings are associated to an area in order to have an approximate location of the different units. Figure 4 shows an example of a partial representation of the numeric state, which is later transformed into a PDDL state. In this case, the units of the player are only located in one area and there is not information available about the enemies yet. Each numerical variable is transformed in a set of logical predicates or numerical functions which are included into the game state depending of the abstraction level. The macromanagement level uses some logical predicates, the numerical functions and a group of special functions which can be tunned in order to change the goal generation process: (i) `(goals-score)` which computes the game score of the plan; (ii) `(total-time)` which computes an approximation of the total time to reach the goals; and (iii) `(max-number-goals)` which defines the maximum number of goals that can be generated. The micromanagement level uses only the logical predicates in order to simplify the plan generation. Figure 4 shows an example of the partial representation of the game state, where there is a *Barracks* building which is built in the first area of the map. Thus, the location of each building is going to be represented using two predicates: (i) `(have_building_barracks 1` which means that there is at least one barracks in the area 1 of the map; and (ii) `(number_building barracks 1)` which means which number of barracks are in the area 1. Besides, there are some other predicates that indicates if the agent can be built `(can_build ?building)` or if the technological level of the agent allows to build that specific building `(tech_level_building ?building ?level)`.

As a result, we have come up with 10 directed actions and 4 derived actions for the micromanagement level. These actions are used by the classical planner to later perform most of the available game behaviors using the goals generated by the macromanagement level. The directed actions (*move a unit, move a units' group, gather, build, train, research, attack, quarter and explore*) produce changes into the environment and the derived actions (*find-unit, find-building-location, find-explore-location and find-attack-location*) collect information to later execute a directed action.

V. Experimental Results

This section presents preliminary results when using the Planning and Execution architecture described in Section IV to play on different scenarios from StarCraft. The Brood War Application Programming Interface (BWAPI) was used to connect the executor to the StarCraft game in order to send and receive real-time information from the game. The experiments were conducted on an Intel Core i7-6700T 2.80 GHZ (32 bits) with 3.50 GB running on Windows 7 for the StartCraft simulator and on and Intel Core i7-6700T 2.80 GHZ (64 bits) with 2 GB running on Ubuntu Linux. The maximum planning time for the low-level planner to solve a problem has been set up to 20 seconds and the horizon `(max-number-goals)` for the high-level planner has been set up to 5 and 3 goals. The high-level planner is OPTIC [2] while the low-level planner is METRIC FF [9]. Each scenario has been played 5 times until the game is finished. Initially we tried to compare our approach with some other approaches, but either they had been developed for other platforms or they needed some kind of previous human-directed learning to work properly.

Table I shows the results of playing on StarCraft with our Multilevel Planning approach. We chose three different scenarios (Astral Balance, Baby Steps, Ice Mountain) where three different strategies (greedy, global, parallel) have been deployed. We have used the Terran race. In general, the Multilevel Planning approach can suggest different set of goals and use them to generate short plans to execute actions in the environment. There are two important aspects to analyze in the results: (i) the planning time; and (ii) the number of planning steps. Besides, we have defined three goal-selection policies in order to analyze the influence of the different goals in the planning and execution process: (a) Greedy goal selection, which generate a plan for each goal; (b) Global goal selection, which generate a plan for all goals; and (c) Parallel goal selection, which extracts a subset of goals that can be reached in parallel. Two goals are considered parallel if the

TABLE I
COMPARING THREE GOAL-SELECTION STRATEGIES OVER THREE DIFFERENT MAPS FROM STARCRAFT. THE PERFORMANCE OF THE DIFFERENT STRATEGIES HAVE BEEN MEASURED OVER SIX METRICS. GOALS CORRESPONDS TO THE NUMBER OF GOALS GENERATED DURING THE MATCH. PLANNING STEPS CORRESPONDS TO THE NUMBER OF PLANNING EPISODES. UNITS CORRESPONDS TO THE NUMBER OF UNITS TRAINED. BUILDINGS CORRESPONDS TO THE NUMBER OF BUILDINGS BUILT. PLANNING TIME CORRESPONDS TO THE AVERAGE TIME OF THE PLANNING EPISODES. TIME CORRESPONDS TO THE TOTAL MATCH TIME.

Map	Strategy	Goals	Planning steps	Units	Buildings	Planning time	Time
Astral Balance	Greedy	18	17	8	4	0.3	5:45
Astral Balance	Global	27	6	18	7	3.55	4:36
Astral Balance	Parallel	25	7	17	6	2.94	5:25
Baby Steps	Greedy	16	15	7	4	0.1	4:38
Baby Steps	Global	31	8	20	6	3.12	5:40
Baby Steps	Parallel	29	7	16	7	1.12	5:15
Ice Mountain	Greedy	15	13	9	3	0.4	3:18
Ice Mountain	Global	28	6	15	8	3.55	4:07
Ice Mountain	Parallel	26	9	17	7	2.34	3:56

game actions from which they are derived can be executed in parallel.

On the one hand, the use of methods that generate a larger goal set (global selection and parallel selection) decreases the number of planning steps but increases the planning time on each iteration. The complexity of the problem to solve is increased by the number of goals. On the other hand, the use of a greedy method that generates a small goal set decreases the planning time but increases the number of planning steps making the agent work slower and decreasing the number of actions per iteration that are executed in the environment.

VI. Related work

The approach described in this paper is focused on applying Automated Planing (AP) as a reasoning model for a StarCraft autonomous player. There are previous approaches that have used either some other AI techniques for reasoning or AP. SOAR [12] is a cognitive architecture that implements state abstractions, planning and multi-tasking based on Finite State Machines (FMS). This architecture was developed using two layers. First layer is a middleware layer that serves as the perception system and gaming interface. Second layer is composed by a set of FSMs that executes parallel actions into the game according to the information obtained from the first layer.

Darmok [16] is an architecture that implements on-line case-based planning by learning from demonstration on WARGUS (an open source clone of WarCraft II). This architecture employs a planning, execution and monitoring loop, similar to the one used in our approach. The execution cycle analyzes if the last action is finished, updates the current state and picks up the next action. However, Darmok must be first trained by playing a collection of games to generate the case library. In [22], authors present a bot that uses a reactive planning implementation of the Goal-Driven Autonomy (GDA) [15]. UAlbertBot [4] uses a heuristic search algorithm to find concurrent plans of actions which are constrained by unit dependencies and resource availability. Theses plans are focused on creating a certain number of units and structures in the shortest possible time span.

The majority of these approaches use control systems for videogames agents, but only one of them plays on Startcraft. Our approach mainly differs from those previous works in the generation and achievement of goals, which is the main focus of this paper. We do not use any training phase to learn.

VII. Discussion and Future Work

In this paper, we propose a Multilevel Planning approach to perform goal-reasoning and planning on RTSVGs using two different levels of planning. This approach has been deployed using the general-purpose software architecture (PELEA) in order to implement an autonomous player for the video-game StarCraft: Brood War. The preliminary results presented in this paper allow us to draw some very interesting conclusions: (i) it is possible to use AP for goal reasoning according to the information about the environment; (ii) divide the complexity of the environment on different levels decreases the complexity of the planning task and the definition of the domain; and (iii) using a temporal planner to generate a sequence of actions, which is going to be transformed into goals, allows to decrease the information used for planning in the second level. Our contribution presents some advantages: it can be easily improved by adding new actions/goals/information on the domain and being easily adapted to different RSTVGs, as the only domain-dependent parts are the Executor module and the planning domains.

While our initial results are encouraging, there is room to improve the approach presented in this paper. On one hand, the different domains used for planning and goal-reasoning can be refined in order to include more actions to generate new goals or more complex plans of actions. Besides, it is possible to define different domains for the other races of the video-game. On the other hand, we can explore other techniques based in Machine Learning to replace both the goal-reasoning or the planning process: (i) learning goal trees to suggest goals; and (ii) using case-based planning to learn previous plan and avoid some planning and replanning episodes. We could also study HTN planning to analyze its flexibility when working on a dynamic environment; it could complement or replace one of the planners.

Acknowledgments

This paper has been partially supported by the European Union's Horizon 2020 Research and Innovation program under Grant Agreement No. 730086 (ERGO) and the Spanish MICINN project TIN2017-88476-C2-2-R.

References

[1] Mitchell Ai-Chang, John L. Bresina, Leonard Charest, Adam Chase, Jennifer Cheng jung Hsu, Ari K. Jónsson, Bob Kanefsky, Paul H. Morris, Kanna Rajan, Jeffrey Yglesias, Brian G. Chafin, William C. Dias, and Pierre F. Maldague. Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[2] J. Benton, Amanda Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, 2012.

[3] Sara Bernardini, Maria Fox, and Derek Long. Planning the behaviour of low-cost quadcopters for surveillance missions. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, Portsmouth, New Hampshire, USA, 2014.

[4] David Churchill and Michael Buro. Build order optimization in starcraft. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2011)*, pages 10–14, 2011.

[5] Marc de la Asunción, Luis A. Castillo, Juan Fernández-Olivares, Óscar García-Pérez, Antonio González, and Francisco Palao. SIADEX: an interactive knowledge-based planner for decision support in forest fire fighting. *Artificial Intelligence Communications*, 18(4):257–268, 2005.

[6] Stefan Edelkamp and Jörg Hoffmann. Pddl 2.2 : The language for the classical part of ipc-4. In *Proceedings of the fourth International Planning Competition. International Conference on Automated Planning and Scheduling*, Whistler, British Columbia, Canada, 2004.

[7] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *JAIR*, 20, 2003.

[8] Javier Garca, Jos E. Flrez, lvaro Torralba Arias de Reyna, Daniel Borrajo, Carlos Linares Lpez, Angel Garca Olaya, and Juan Senz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operational Research*, 227(1):216–226, 2013.

[9] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[10] Benjamin Johnson and Mark Roberts. Goal reasoning with informative expectations. In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems (ACS-16)*, 2016.

[11] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence Journal*, 42(2-3):189–211, 1990.

[12] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, sep 1987.

[13] Moisés Martínez, Fernando Fernández, and Daniel Borrajo. Planning and execution through variable resolution planning. *Journal of Robotics and Autonomous Systems*, In press.

[14] Matthew Molineaux, Matthew Klenk, and David W. Aha. Goal-driven autonomy in a navy strategy simulation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[15] Héctor Muñoz Avila, Ulit Jaidee, David W. Aha, and Elizabeth Carter. Goal-driven autonomy with case-based reasoning. In *Proceedings of the 18th International Conference on Case-Based Reasoning Research and Development*, ICCBR'10, pages 228–241, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In *Proceedings of the 7th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ICCBR '07, pages 164–178, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] Alison Paredes and Wheeler Ruml. Goal reasoning as multilevel planning. In *Proceedings of the ICAPS-17 Workshop on Integrated Execution of Planning and Acting (IntEx-17)*, 2017.

[18] Alberto Pozanco, Susana Fernandez, and Daniel Borrajo. Urban traffic control assisted by ai planning and relational learning. In *Proceedings of the 9th International Workshop on Agents in Traffic and Transportation (IJCAI'16)*, 2016.

[19] Ezequiel Quintero, Vidal Alcázar, Daniel Borrajo, Juan Fernández-Olivares, Fernando Fernández, Angel García Olaya, César Guzmán, Eva Onaindia, and David Prior. Autonomous mobile robot control and learning with the pelea architecture. In *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*, San Francisco, CA, USA, 2011.

[20] K. Rajan, C. McGann, F. Py, and H. Thomas. Robust mission planning using deliberative autonomy for autonomous underwater vehicles. In *Proceedings of the Workshop on Robotics in Challenging and Hazardous Environments, ICRA*, Rome, Italy, 2007.

[21] Mark Roberts, Vikas Shivashanka, Ron Alford, Michael Leece, Shubham Gupta, and David W. Aha. Goal reasoning, planning, and acting with actorsim, the actor simulator. In *Proceedings of the Fourth Annual Conference on Advances in Cognitive Systems (ACS-16)*, 2016.

[22] Ben Weber, Michael Mateas, and Arnav Jhala. Building human-level ai for real-time strategy game. In *Proceedings of AIIDE Fall Symposium on Advances in Cognitive Systems*, 2011.