A Knowledge Based Process for the Generation of HTN Domains from VGDL Video Game Descriptions

Ignacio Vellido, Juan Fdez-Olivares, Raúl Pérez
Dpto. Ciencias de la Computación e I.A
University of Granada, Spain
ignaciove@correo.ugr.es
{faro, fgr}@decsai.ugr.es

Abstract

This work addresses the problem of how to automatically generate Hierarchical Task Network (HTN) planning domains, in order to faithfully represent the dynamics of a video game. We introduce a KE process that receives as input a video game and game-level descriptions in the standard language VGDL, and returns an HTN planning domain and problem. Therefore, with the inclusion of a agent strategy, an HTN planner can guide the behaviour of an automated player. The effectiveness of this process is tested in 4 different video games of the GVGAI environment, a standard framework containing more than 100 different video games descriptions, and whose main advantage in our approach is the ability to integrate an HTN planner as a controller in potentially any of its video games, requiring a minimum effort from a knowledge engineer.

Introduction

The use of automated planning (and particularly HTN planning) to guide the behaviour of either an automated player or Non Playable Characters (NPCs) in a video game has been previously addressed (Kelly et al. 2008; Hoang, Lee-Urban, and Muñoz-Avila 2005) providing several advantages, among them that the agent is endowed with deliberative reasoning capable of solving problems in the video game, thus increasing the cognitive capabilities of automated players.

HTNs are hand-coded structures that encode knowledge about the domain. A planner takes as input an HTN domain and information about the specific planning problem, and outputs a plan that guides the agent behaviour in the video game. However, an important obstacle hinders the adoption of this technology: the creation of a planning domain that perfectly fits to the requirements of a video game is a long, difficult and meticulous task, where a minimum mistake can break the domain and produce inconsistent plans. For this reason we propose an automated process that starts from a formal description of the video game, and ends with an HTN domain that incorporates the objects and dynamics of the game described in the initial format, allowing the introduction of a planner to solve problems in the video game.

We are using a simple but expressive language such as VGDL (Video Game Description Language) (Schaul 2013) to formally describe the objects and dynamics of the video game, as well as the different levels or scenarios. This is a language used to describe many video games. For example, more than 100 video games are defined in the GVGAI environment (Perez-Liebana et al. 2016) following the VGDL standard. Our aim is to seamlessly integrate and HTN planner into the GVGAI framework (Perez-Liebana et al. 2015) while reducing at the minimum the KE effort to represent planning knowledge. The goal of GVGAI is to serve as a benchmark for testing techniques oriented to Artificial General Intelligence (AGI), and this automated KE process enables HTN planning techniques to be tested and compared with other AGI techniques that have already been successfully proved on previous competitions over this environment (Torrado et al. 2018). In this work we are presenting the first step towards this goal.

Therefore, our main contribution is an automated knowledge based process that, receiving a VGDL game and level description, produces both an HTN domain that represents the game objects, their relationships and the dynamics, and a problem representing a concrete initial game configuration. The so generated domain and problem can be provided later as input to an HTN planner to produce plans that guide the behaviour of the agent. Moreover, having the generated domain as a basis, a human designer can also define a specific agent strategy to act accordingly towards the desired goals[1].

The effectiveness of this process was tested in four cases of study, producing domains for two puzzle games and two reactive games. We have also developed a basic replanning strategy and finally integrated the planner as a controller for the GVGAI environment, and improved the default agent strategy to visualize with GVGAI the potential on reaching goals in the tested video games.

In the following sections we introduce some background concepts on VGDL and the HTN language used, describe the knowledge process addressed and finish analysing the results obtained with the experiments.

[1]Hence we are still far from AGI, but we point out in the conclusions that machine learning techniques could be built upon our proposal.

```
SpriteSet
   user    > VerticalAvatar
   boulder > Missile        orientation=DOWN
```

(a) Definition of the objects and their parameters.

```
LevelMapping
   u > user
   b > boulder
```

(b) Characters that represents each object in the level definition file.

```
InteractionSet
   user boulder > killIfFromAbove
```

(c) An example of an interaction between the objects

```
TerminationSet
   SpriteCounter stype=user limit=0 win=False
```

(d) An ending criteria for the game.

Figure 1: The four parts that compose a VGDL game description.

Related Work

The generation of planning domains from descriptions represented in domain-specific languages has been addressed previously in approaches like (Castillo et al. 2010) for eLearning scheduling, (González-Ferrer, Fernández-Olivares, and Castillo 2013) for Business Process Management, and (Fdez-Olivares et al. 2011) for supporting clinical processes. All these works use a language related to a concrete problem and generate planning domains that support expert decision making.

Regarding video games, automated planning has for long been considered an enabling technology to control the behaviour of deliberative agents. (Černỳ et al. 2016) shows an in-depth review of related work about planning and video games. Nevertheless, up to the authors' knowledge, the works are focused on evaluating the performance of planners, and planning domains are manually represented. In most of the cases, only a single planning domain is represented and planner performance is evaluated on different problems (or game scenarios/levels). On the other hand, approaches like (Geffner and Geffner 2015) go further, and show that planning algorithms are competitive with respect to standard techniques in Atari video games, but without a compact PDDL-model for those games. Regarding video game frameworks, several planning architectures have been integrated with standard frameworks like StarCraft (Martínez and Luis 2018) or Minecraft (Bonanno et al. 2016), based on the idea of evaluating the performance of an online planning architecture using a single planning domain.

It is widely known that machine learning techniques have proven to be useful when applied to video games. Concerning GVGAI, some approaches provide results that allow reactive controllers to learn with relative success (Torrado et al. 2018). However, these controllers lack deliberative capabilities. On the other hand, in this paper we are interested in HTN approaches, which are well suited to providing knowledge to deliberative agents, although a knowledge engineer needs to provide advice to obtain an effective controller. For these cases approaches like (Gopalakrishnan 2017) and (Zhuo et al. 2009) propose different methodologies to learn HTN domain structures automatically. We point out in the last section a research direction to address this issue.

As previously mentioned, we are concerned with the video game framework GVGAI (Perez-Liebana et al. 2016) (described later). To the best of our knowledge, the only attempt of applying planning to this framework is (Couto Carrasco 2015), where authors manually define a controller for puzzle games using PDDL. The approach here presented tries to go one step further, by proposing a methodology to automatically generate HTN planning domains from *any* video game described in VGDL in this framework, and integrating an HTN planner in order to control the behaviour of an automated player.

Background

GVGAI and VGDL

GVGAI (General Video Game AI) (Perez-Liebana et al. 2016) is a framework for evaluating the performance and generalization capabilities of AI-based techniques in multiple domains (video games), and includes a vast variety of video games descriptions and levels of all kind. Although the framework is oriented to address problems of General Artificial Intelligence, we aim to use this framework as a workbench for planning techniques, specially for the integration of planning and acting.

The underlying language that GVGAI uses for video games descriptions is a variant of VGDL (Video Game Description Language) (Schaul 2013), a high-level language to describe video games, focused on speed and simplicity. There is a huge variety of predefined types in VGDL, ranging from static ones to different classes of NPCs and agents, making possible to define many types of video games.

A VGDL game is specified with two text files, one detailing the types of objects, their relationships and dynamics (a video game description file), and another describing the game level, representing an initial configuration for the objects involved in the game. A VGDL description is structured in four parts (see Figures 1a, 1b, 1c and 1d):

- **SpriteSet**: A specification of game objects (avatar, enemies, walls, missiles, etc.). Figure 1a shows an example of object definition and their attributes, where we can see that *boulder* is a missile (an object with a continuous movement) that only goes downward. VGDL incorporates a set of default avatars (according to the different types of available actions that an avatar can execute), for

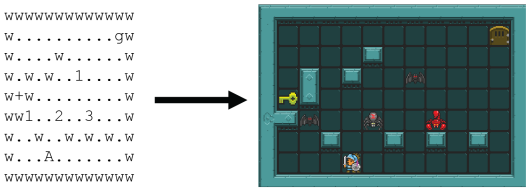

Figure 2: Level representation in GVGAI. Notice how each character represents an instance of an object. For example: "*w*" means *wall*, "." means a free cell, "*A*" means *avatar* and each number a different enemy.

example, the same figure shows that *user* is an instance of *VerticalAvatar*, i.e., a player agent with two available movements, *up* and *down*. The declaration of game objects can also be hierarchical, allowing the inheritance of attributes and behaviour.

- **LevelMapping**: A set of characters used in a level file to represent the objects that appears in a game level scenario. Figure 1b shows how the *boulder* and *user* objects are respectively represented as *b* and *u* in the level description.
- **InteractionSet**: A specification of the interactions between the objects, indicating which actions have to be executed over objects (by the game engine) when they collide with each other. In the same way as sprites, there is a previously defined repertory of interactions in VDGL. In Figure 1c we can see an interaction between *boulder* and *user*, called *killIfFromAbove*, meaning that the boulder "kills" the avatar user if the former falls from above the latter. The order upon which the objects are defined is relevant, being the first one the producer of the action and the second one the receiver of the effects.
- **TerminationSet**: A list of criteria that makes the game finalize. We can see in Figure 1d an example, stating that when the number of *users* alive in the game reaches zero, the game ends and the agent loses.

A game level file is composed of a 2D matrix of characters where each cell indicates the starting position of an object instance. Each character is associated with its object as stated in the *LevelMapping* section. Figure 2 shows the transformation from the level description to the visual representation in GVGAI.

We can see that the video game description and its different levels follows a similar conceptualization to the HTN domain representation and its associated problems. Furthermore, the turn-based cycle characteristic of tile-based video games reassembles a recursive HTN-task with termination criteria. These similarities will be exploited in the section devoted to the methodology.

HPDL

Regarding knowledge representation and reasoning, we are using HPDL and SIADEX, a planning language and a planner designed to represent HTN problems successfully applied to several application domains (Fdez-Olivares et al. 2006; Fdez-Olivares et al. 2019). The syntax of HPDL embodies the syntax of PDDL (Fox and Long 2003) 2.1 level

```
(:task <T>
    :parameters (...)
    (:method <T_1>
        :precondition (...)
        :tasks (
            (subtask_1 ...)
            ...
            (subtask_n ...)
        )
    )
    ...
    (:method <T_N>
        ...
    )
)
```

Figure 3: Task representation in HPDL. Each task receives a list of parameters and contains one or more methods. Similarly to a classic PDDL primitive, each method contains a section of preconditions and effects, the latter represented as abstract subtasks.

3 (the *de facto* standard for planning domains) to represent PDDL objects, types, predicates and primitive actions. Further, it extends PDDL to represent tasks at different levels of abstraction and methods to describe alternative decomposition schemes for compound tasks (see Figure 3). Each task can be decomposed into several, and each decomposition alternative is described by a decomposition method. Each method contains a precondition that describes its applicability condition. An HPDL problem looks the same as in PDDL, except for the goal. In HPDL the goal is not a set of facts that needs to be true at the end of the plan, but a set of tasks to be accomplished.

Methodology

The automated generation process starts from two input files describing, respectively, a VGDL video game and a specific level of that game. As a result, an HPDL domain and a problem are produced. The procedure, represented in Figure 4, consist of the following steps:

1. **Extraction of Game Entities**. The VGDL game file is parsed[2] and a set of game entities are extracted, each corresponding to the *SpriteSet, InteractionSet*, and *LevelMapping* sections of the video game description[3].
2. **Domain Generation**. These structures enter in a module responsible of producing the output domain, using also as input previously defined templates stored in a Knowledge Base. Based on the parsed types of entities and interactions, this module produces different sections of an HPDL domain (types, predicates, tasks, methods and primitives).

[2]Using a parsing process based on the ANTLR (Parr and Quong 1995) parsing facilities.

[3]The *TerminationSet* part is not automatically processed due to its complexity and ambiguous semantics. Nevertheless, the termination conditions of the game are manually represented in HPDL

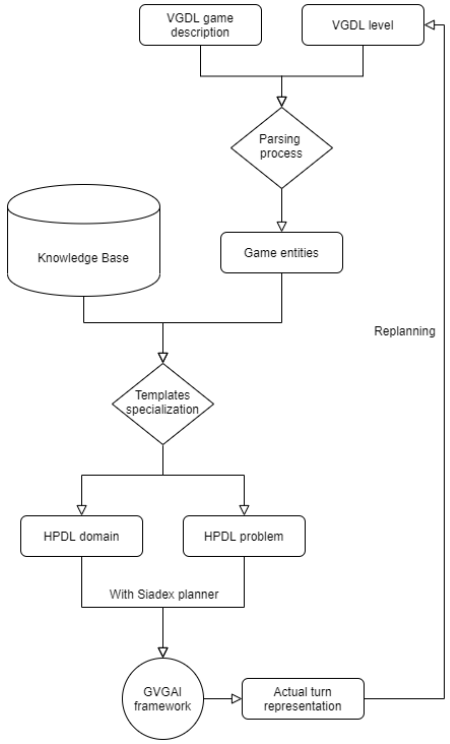

Figure 4: General parsing process for a video game.

3. **Problem Generation**. In a similar way, the structures parsed are provided as input, along with templates in the Knowledge Base, to a module in charge of producing the HPDL problem (objects, init state and goal).
4. **Planning and execution**. The planner uses both the domain and the problem generated to produce a plan for GVGAI, where a basic replanning strategy has been defined to integrate an online planning and execution process in order to control the behaviour of the avatar representing the automated player. The online execution process is designed to cope with non-deterministic situations, since the domains produced are deterministic.

Game entities

The game entities represent the relevant concepts of a VGDL game description and are conveniently translated into objects of the HTN domain. Different information is extracted from each part of a VGDL description file:

- **SpriteSet**: The name and type of each object are parsed from this section. In addition, because the objects are defined hierarchically in the VGDL description, it is necessary to maintain this hierarchy between entities in the HPDL domain. Lastly, the functional parameters (like speed or movement directions, not the ones concerning the visual representation) are also parsed to specify the properties of the objects.
- **InteractionSet**: The type and the objects involved in each interaction are parsed and stored, as well as the role of each object of the pair, either *producer* or *receiver* of the action representing the interaction.
- **LevelMapping**: The correspondences between the characters in the level description and the specific instances of the game objects are parsed in order to be able to produce the HPDL problem.

```
(:action MISSILE_MOVE
    :parameters ()
    :precondition ()
    :effect (
        forall (?m - Missile) (and
            (when (orientation-up ?m)
                (and
                (assign (last_coordinate_y ?m)
                        (coordinate_y ?m) )
                (decrease (coordinate_y ?m) 1)
                )
            )

            (when (orientation-down ?m)
                ...
            )
            ...
        )
    )
)
```

Figure 5: Automatically generated primitive for a movable object, updating the state of all the sprites of type *Missile*. In this case, the action checks the orientation of the instances and sets their position accordingly.

Knowledge base

The Knowledge Base contains templates used to produce the tasks, predicates, methods and primitives that forms an HPDL domain, as well as the initial instantiated predicates included in a HPDL problem. These templates are game-independent, and only related to the specific element they represent. Therefore, the Knowledge Base is organised in three sections:

- **Sprites**: This section of the Knowledge Base is concerned with the representation of objects in the game (not including the avatar). Each *sprite* results in a different *object* on HPDL and, depending on its type, additional primitives and predicates are included in the Knowledge Base. Furthermore, every *sprite* has HTN-functions (similarly to PDDL-functions) to represent its current and previous position. When treating with a movable object, primitives for the direction are also represented.

  As an example, a *Missile* type object will produce primitives and predicates to update its movement in each game turn, moving only in the direction indicated by its parameters, as Figure 5 shows.
- **Avatars**: This section is concerned with a subset of sprites representing the automated players in the game. The actions represented for avatars in the Knowledge Base are

```
(:action AVATAR_MOVE_UP
    :parameters (?a - <T>)
    :precondition (and
        (can-move-up ?a)
        (orientation-up ?a)
    )
    :effect (and
        (decrease (coordinate_x ?a) 1)
    )
)
```

Figure 6: Example of an action template for an avatar movement, where *T* indicates the avatar type.

```
(:action MOVING_WALL_STEPBACK
    :parameters (?x - moving ?y - Inmovable)
    :precondition (and
      (= (coordinate_x ?x) (coordinate_x ?y))
      (= (coordinate_y ?x) (coordinate_y ?y))
    )
    :effect (and
      (assign (coordinate_x ?x) (lastCoordinate_x ?x))
      (assign (coordinate_y ?x) (lastCoordinate_y ?x))
    )
)
```

Figure 7: Example of an interaction, called *stepBack*, involving a *moving* type object and a *wall* one, and resulting in the *moving* object going back to its previous position. It uses predicates to keep track of the actual and previous object coordinates.

limited for tile-based games with grid physics, usually including moves in the four cardinal directions and the possibility of using as a resource a previously defined *sprite* (for example, a sword that can kill enemies).

The Knowledge Base stores templates for each possible action of a VGDL avatar and knows which ones are available for each type. With these templates, and depending on the type of avatar, a simple strategy is included in the domain, although we encourage to manually define one focused on the objectives the agent is supposed to attain.

Figure 6 shows an example of an action template. After the parsing process where the avatar type is extracted, the variable *T* will be instantiated and the primitive included in the output domain.

- **Interactions**: This section stores templates for VGDL interactions, representing the collision of two objects in the same tile. Each one has particular primitive and method templates, with the resulting instantiation as shown in Figures 7 and 8. The method acts as a wrapper including extra verifications and better organisation in the domains.

```
(:method moving_wall_stepback
   :precondition (and
     (not (= (coordinate_x ?x) -1))
     (not (= (coordinate_x ?y) -1))
     ...
   )
   :tasks (
     (MOVING_WALL_STEPBACK ?x ?y)
     ...
   )
)
```

Figure 8: Method wrapper for the interaction in Figure 7. We include verifications of the objects position (-1 indicates they are not actually present in the level) to reduce the planner search time.

Generating the output

Having the game entities and the templates returned from the Knowledge Base, it is necessary to make an instantiation before outputting the HPDL domain and problem. With this idea in mind, the output is divided by each file:

Domain generation Based on the templates from the Knowledge Base and the game entities from the parsing process, the domain is constructed as follows:

- **HPDL types**: For each child sprite, the hierarchy is reflected defining the parents as the object type. When reached a sprite without parent, its type defined in the VGDL file will be its HPDL type. All sprite types will inherit from the most generic object, in the HPDL case, a supertype called *Object*. This supertype allows us to generalize behaviour in predicates and functions.

  Figure 9 shows an example of a hierarchy in one of the resulting domains. Every sprite (represented enclosed in a elipsis) inherits from its type, while these types inherit from the supertype Object.

- **HPDL predicates and functions**: It is necessary to keep two types of predicates: to know the orientation of all movable objects and to control when a movement is available for the agent.

  As for the functions, it is needed to keep track of the former and actual position of each object (avatars and sprites). This is necessary because there are common interaction, like *stepBack*, that returns an object to its previous position when it collides with another. Additionally, it is important to maintain a counter of each type of object defined and the resources the avatar has obtained in order to keep track of the possible ending criteria.

- **HPDL compound actions**: A global task *Turn* is generated to represent the change of state of all the entities in a game-turn. Roughly speaking, this task represents the game forward model, thus enabling the planner the ability to reason about the game dynamics. This task is composed of two methods: one checking the ending criteria

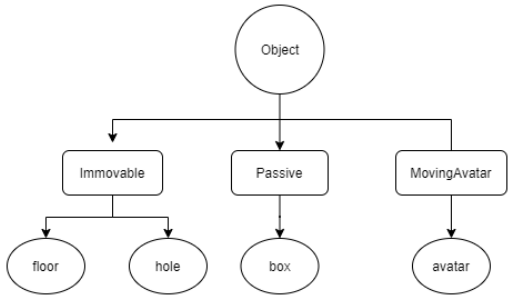

Figure 9: Representation of the game object hierarchy in an output domain for the game Sokoban.

```
(:task Turn
    :parameters ()
    (:method finish_game_1
        :precondition (= (turn) 10)
        :tasks ()
    )
    (:method finish_game_2
        :precondition (= (counter_avatar) 0)
        :tasks ()
    )
    ...
```

Figure 10: Example of ending criteria for a domain. If any of the preconditions evaluate as true, the planner stops and return the actual plan.

and another invoking a series of subtasks with the specific game representation (further detailed in Subsection **HPDL Turn**). These subtasks consist of:

- A compound subtask that represents the avatar movements.
- A subtask with the changes of state of non-static sprites (movable object like bullets or missiles).
- A recursive task that checks all possible interactions in the game.

- **HPDL primitives**: The generation of primitives is structured as follows:
  - For each possible movement of the avatar an action is generated, plus one for the no-movement.
  - For the non-static sprites, a primitive representing the update of state is produced, as previously shown in Figure 5 for a *Missile* type object.
  - Finally, for each interaction a primitive is included to reproduce its effects, as in Figure 7. The precondition primarily checks the collision of objects, but depending on the type of interaction additional verifications may be included, like checking the orientation of the sprites.

Problem generation Similarly to the domain generation process, the problem is constructed with the help of the game entities and the Knowledge Base. For each character on the level description file, a new instance is created indicating its position at the start of the game, and the counters are appropriately initialized. The goal will be the call to the recursive task *Turn*, where the ending criteria and the game cycle is defined. Figure 10 shows examples of two ending conditions used in the experimentation.

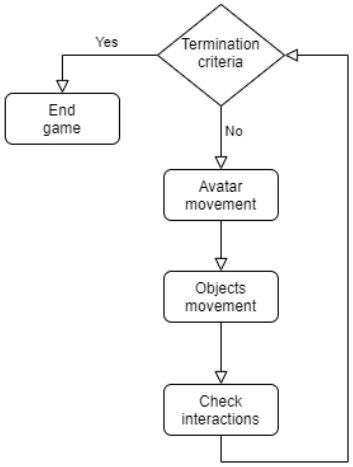

Figure 11: Turn cycle represented in the domains.

HPDL Turn

In classical (non HTN) the goal is a state description that has to be satisfied by a sequence of actions found by a search process, e.g. *the avatar must be on the exit door*. However, in HTN the goal is not a final state, but a set of tasks to be decomposed. Therefore, the task-goal in our HPDL problem consist only of a recursive task called *Turn*. This task provides additional knowledge to the planner about the game representation. However, because we tried to keep the process as general as possible, we did not include game-specific heuristic knowledge, and we considered the use of techniques to automatically extract it as part of Future Work. Specifically, this *Turn* task codifies the changes produced by avatars, sprites and interactions in a game cycle.

As shown in Figure 11, is divided in four subtasks:

- At the beginning of each turn, the termination criteria are checked. In case they are achieved, the planner stops with the current plan.

  As we are only focusing on the representation of the dynamics, and not in winning the game, we used two conditions to determine the end of the planner search: a fixed number of calls to the task *Turn*; or when the quantity of avatars alive reached zero.

- Otherwise, a new cycle starts with the following procedure:
  1. A movement for the avatar is selected. By default it is chosen the first available action for the agent.
  2. The rest of objects update their situation.
  3. For each pair of objects it is checked if an interaction has been produced, invoking the corresponding primitive if true and updating the planning state as necessary.

	Syntax elements							Semantic elements			
	Primitives	Tasks	Methods	Types	Supertypes	Predicates	Functions	Determinism	Movable objects	Agent actions	Interactions
Sokoban	14	5	20	5	3	10	14	Yes	No	4	5
Brainman	33	5	38	11	5	11	22	Yes	Yes	4	23
Aliens	17	5	19	10	9	9	25	No	Yes	3	9
Boulderdash	28	5	33	10	9	12	27	No	Yes	5	17

Figure 12: Syntax and semantic elements of the four domains produced.

Replanning

For replanning, we integrate the parsing process as an agent in the GVGAI framework that updates the HTN problem wherever there are no more remaining movements for the avatar. This is simple because the representation of the actual state of the game in GVGAI follows a similar conceptualization as in VGDL.

With this process we are not only reducing the planning computation time (because we no longer require to plan for a great number of turns), but also helping the agent to face non-deterministic adversities.

Experiments

It is important to note that we are only focused on extracting the video game dynamics in order to be handled by an HTN planner. We have intentionally obviated the generation of heuristic knowledge to come up with plans that solve the videogames. Because of that, the experimentation focuses on validating the HTN representation of the game dynamics, not in testing the quality of the plans.

We experimented with four different games, two puzzles games and two reactive games, containing different levels of determinism and non-determinism, of gradually increasing representation complexity. These games are:

- **Sokoban.** A puzzle game where the agent has to push all boxes into specific tiles on the map. The avatar cannot grab any box and must push it in the same direction it is facing. It features no NPCs or any kind of non-determinism, making Sokoban the most simple game from the ones we tested.
- **Brainman.** Another puzzle game consisting on making the agent go to the exit. As in Sokoban, in his way he must push the keys to open the doors. However, the keys act as missiles, meaning that when one is pushed it follows the same direction until it collides with something.
- **Aliens.** Adaptation of the classic arcade game, where the agent must kill all the aliens that spawn above him. In this version of the game the aliens move randomly to the sides and throw bombs to the avatar, making it highly non-deterministic and thus difficult to represent.
- **Boulderdash.** The objective is to collect nine gems and go to the exit. There are also enemies and falling boulders around the map to disturb the agent during his task. In occasions it is needed to use a shovel to make way through the level, but at the same time taking care not to make the boulders fall in undesired places.

  Boulderdash is characterized by being strongly non-deterministic, multi-objective (in the sense that the avatar has to reach several subgoals and determine the order to achieve each one of them) and sometimes unsolvable, making it by far the most complex game of the four.

  In addition to the same kind of experiments we made with the others games, for *Boulderdash* we manually defined an agent strategy to show the feasibility of the domain produced.

```
:action (AVATAR_MOVE_UP A0)                                  ┐
:action (BOULDER_MOVE)                                       │
:action (DIAMOND_AVATAR_COLLECTRESOURCE x3 A0)               ├ First turn
:action (BOULDER_DIRT_STEPBACK o2 w25)                       │
:action (BOULDER_DIAMOND_STEPBACK o0 x1)                     ┘

:action (AVATAR_MOVE_UP A0)                                  ┐
:action (BOULDER_MOVE)                                       │
:action (DIRT_AVATAR_KILLSPRITE z17 A0)                      ├ Second turn
:action (BOULDER_DIRT_STEPBACK o2 w25)                       │
:action (BOULDER_DIAMOND_STEPBACK o0 x1)                     ┘

…
```

Figure 13: Portion of a plan returned from the Siadex planner, representing two turns in the *Boulderdash* game. The first two primitives of each turn in this plan indicates the avatar and object movements, and the following three the interactions produced in the turn.

All of the games were treated with the same procedure: the group of templates representing the elements in the game were included in the Knowledge Base, and a compiling process was made from VGDL to an HPDL domain, consisting of a description of the game dynamics in form of primitives, tasks, methods, etc., and a naive agent strategy. This strategy chooses the first available move for the agent, without considering any objective. The next step for each game was to test it in multiple levels with the Siadex planner and the GVGAI framework.

As shown in Figure 13, the planner returns a serie of primitives including the avatar and the rest of sprites movements, and an action for each interaction produced in the turn. To transform this plan into GVGAI actions only the ones representing the avatar movement are needed, because the rest merely update the planning state in each turn.

To validate the domains, we followed a procedure where we followed the game execution in the planner state, verifying that the game dynamics were correctly represented. As an example, in Boulderdash we knew a boulder should go down until it collides with a solid object. Therefore, we visually corroborated that the boulder coordinates were accurately updated in the planner state, and the collision interaction was called whenever the boulder impacted with an object.

Further, with the GVGAI integration we checked that the

	Turns	Actions	Expansions	Time (s)
Sokoban	10	14	2075	0.00999
	20	24	3945	0.01748
	50	54	9555	0.04111
Brainman	10	21	6049	2.08517
	20	41	12279	3.49277
	50	101	30969	8.73218
Aliens	10	50	411	0.00499
	20	100	821	0.01111
	50	250	2051	0.02489
Boulderdash	10	126	844178	36.4963
	20	196	1244760	55.2729
	50	257	1737375	72.6214

Figure 14: Details of executions of the different games. Each one was tested for 10, 20 and 50 turns, using the Siadex planner. Siadex uses a state-based forward search process (using DFS). Actions includes avatar, sprites and interactions changes of state. Expansions represent the number of times a method-decomposition process is applied in the search process.

outputted plans could fit appropriately within the game dynamics.

Figure 12 shows details of the domains produced, including the dimensions of each domain in number of tasks, predicates, etc. As well as the semantic characteristics of the games. Additionally, in Figure 14 we show the results of actions, expansions and the duration of different executions in each domain, for levels of similar complexity between games, precisely the first level of each game in the GVGAI framework.

From the tables we see that when the complexity of the game increases (seen in the number of interactions and types) the dimensions of its domain equally augments. Moreover, the bigger the number of interactions and objects defined, the bigger the dimensions of the domain produced.

Because non determinism was not represented in the domains, we see that puzzle games can be in some cases more computationally expensive than reactive ones. This shows us the drawback of not including non-determinism, but the dynamic events in these games makes large plans useless, and a reactive approach with replanning can produce much better results during execution.

Additionally, we created a handmade strategy for the Boulderdash game, showing us the feasibility of the domain in resolving subgoals. This strategy tries to attain the nearest gem for the agent, evading enemies and falling boulders when necessary, as shown in Figure 15.

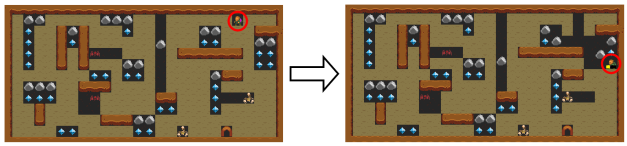

Figure 15: With the inclusion of a handmade strategy in the outputted domain, the agent is capable of attaining subgoals in *Boulderdash*. In this case, we can see the agent in the starting position of the level and after obtaining the four nearest gems.

Conclusions and Future Work

In this paper we have presented an automated knowledge engineering process to generate a huge variety of HTN planning domains for video games. We have shown experimental results on 4 video games represented in the video game description language VGDL. Potentially, we are able to automatically generate as many HTN domains as contained in the video games repertory of the GVGAI framework (at present, more than one hundred). The process requires to represent behaviour patterns of avatars and objects in a Knowledge Base which are latter used as input by a compiling process, generating domains and problems ready to be used by an HTN planner. We estimate that we can save weeks of work for a knowledge engineer devoted to design planning-based deliberative agents to act on these video games.

However, we feel that this project still has some points that can be improved upon and be regarded as future work:

- *Representation of non-determinism*. This kind of situations appear frequently not only in video games, but also in real life problems. In some cases a continuous planning strategy integrating replanning with reactive behaviour can be enough to face this problem. Although if we aim to generate a flexible and self-contained domain, we think that the best approach is the introduction of non-deterministic planning approaches, as (Kuter et al. 2009).
- *A more complex agent strategy*. Even though we are capable of representing the dynamics of the games, we are facing the problem of integrating a learnt agent strategy, to solve video game goals, into the HTN domain generated by the knowledge based process here presented.

  This is not an easy task, because we are no longer considering an unique parsing process. It is necessary to analyse the game and comprehend it. At present we are addressing the automated discovery of game strategies in two different lines, the techniques proposed in (Segura-Muros, Pérez, and Fernández-Olivares 2017) and the subgoal selecting architecture of (Núñez Molina, Fdez-Olivares, and Pérez 2020).

Furthermore, the quality of the results are affected by multiple factors, mainly by the planning language and the planner used. VGDL and HPDL are just proposed as candidates to experiment with the GVGAI framework. This process can be generalized for others HTN languages and other descriptions of objects and interactions that keep similar characteristic with the ones aforementioned. For our next steps we are focused on producing domains in languages as HDDL (Höller et al. 2019) and SHOP (Nau et al. 2003).

Acknowledgements

This research is being developed and partially funded by the Spanish MINECO R&D Project TIN2015-71618-R and RTI2018-098460-B-I00.

References

Bonanno, D.; Roberts, M.; Smith, L.; and Aha, D. W. 2016. Selecting subgoals using deep learning in minecraft: A preliminary report. In *IJCAI Workshop on Deep Learning for Artificial Intelligence*.

Castillo, L.; Morales, L.; González-Ferrer, A.; Fdez-Olivares, J.; Borrajo, D.; and Onaindia, E. 2010. Automatic generation of temporal planning domains for e-learning problems. *Journal of Scheduling* 13:347–362.

Černỳ, M.; Barták, R.; Brom, C.; and Gemrot, J. 2016. To plan or to simply react? an experimental study of action planning in a game environment. *Computational Intelligence* 32(4):668–710.

Couto Carrasco, M. 2015. Creació d'un controlador automàtic pel concurs gvg-ai. http://hdl.handle.net/10230/25487.

Fdez-Olivares, J.; Castillo, L.; García-Pérez, Ó.; and Palao, F. 2006. Bringing users and planning technology together. experiences in siadex. In *Proceedings ICAPS*, 11–20.

Fdez-Olivares, J.; Castillo, L.; Cózar, J. A.; and García Pérez, O. 2011. Supporting clinical processes and decisions by hierarchical planning and scheduling. *Computational Intelligence* 27(1):103–122.

Fdez-Olivares, J.; Onaindia, E.; Castillo, L. A.; Jordán, J.; and Cózar, J. A. 2019. Personalized conciliation of clinical guidelines for comorbid patients through multi-agent planning. *Artif. Intell. Medicine* 96:167–186.

Fox, M., and Long, D. 2003. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research* 20:61–124.

Geffner, T., and Geffner, H. 2015. Width-based planning for general video-game playing. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

González-Ferrer, A.; Fernández-Olivares, J.; and Castillo, L. 2013. From business process models to hierarchical task network planning domains. *The Knowledge Engineering Review* 28(2):175–193.

Gopalakrishnan, S. 2017. Learning hierarchical task networks using semantic word embeddings.

Hoang, H.; Lee-Urban, S.; and Muñoz-Avila, H. 2005. Hierarchical plan representations for encoding strategic game ai. In *AIIDE*, 63–68.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2019. Hddl – a language to describe hierarchical planning problems.

Kelly, J. P.; Botea, A.; Koenig, S.; et al. 2008. Offline planning with hierarchical task networks in video games. In *AIIDE*, 60–65.

Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2009. Task decomposition on abstract states, for planning under nondeterminism. *Artificial Intelligence* 173:669–695.

Martínez, M., and Luis, N. 2018. Goal-reasoning in starcraft: brood war through multilevel planning. In *XVIII Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2018) 23-26 de octubre de 2018 Granada, España*, 107–113. Asociación Española para la Inteligencia Artificial (AEPIA).

Nau, D. S.; Au, T. C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Núñez Molina, C.; Fdez-Olivares, J.; and Pérez, R. 2020. Improving online planning and execution by selecting goals with deep q-learning. ICAPS 2020 Workshop on Integrated Execution and Goal Reasoning.

Parr, T. J., and Quong, R. W. 1995. Antlr: A predicated-ll(k) parser generator. *Software: Practice and Experience* 25(7):789–810.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8(3):229–243.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.; and Thompson, T. 2016. The 2014 general video game playing competition.

Schaul, T. 2013. A video game description language for model-based or interactive learning. 1–8.

Segura-Muros, J. A.; Pérez, R.; and Fernández-Olivares, J. 2017. Learning htn domains using process mining and data mining techniques.

Torrado, R. R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep reinforcement learning for general video game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Zhuo, H. H.; Hu, D.; Hogg, C.; Yang, Q.; and Muñoz-Avila, H. 2009. Learning htn method preconditions and action models from partial observations. 1804–1810.