Chapter 5 - Learning in GVGAI

Jialin Liu

The previous chapters mainly discuss the design of the GVGAI framework and planning in GVGAI, in which a Forward Model (FM) of each game is available. GVGAI can also be used as a Reinforcement Learning (RL) platform. In this context, the agent is not allowed to interact with the FM to plan ahead actions to execute in the real game, but need to learn through experience, thus playing the actual game multiple times (as *episodes* in RL), aiming at improving their performance progressively.

Two research questions are conducted. First, can we design a learning agent A_1 which is capable of playing well on unknown levels of game G_1 after having been trained on a few known levels of the same game? Then, on a new game G_2, is it possible to train an agent A_2, based on A_1, more efficiently than from scratch? The first question focus on improving an agent's ability of performing similar tasks with same objectives and rules. An application is the autonomous order picking robots in the warehouses, which can optimize the routes and travel through different picking areas to maximize pick-up efficiency. The second one aims at enhancing learning efficiency on new tasks by transferring the knowledge learnt on distinct ones (e.g., tasks with different rules). For instance, a tennis player perhaps masters badminton more quickly than someone who plays neither sport.

This chapter raises first the challenges of learning in GVGAI (Section 2.1). Section 2 provides an overall view of the GVGAI learning platform, then present the two environments that have been implemented. Section 3 presents the rules and games used in the competitions, the approaches that have been used by the submitted agents to the competitions, together with the analysis of the performance of the different approaches. The competition entries are described in Section 4.

1 Challenges of Learning in GVGAI

The planning and learning tasks share some identical challenges including, but not limited to, the lack of a priori knowledge, requirement of generality, delayed rewards (usually together with a flat reward landscape) and real-time constraints.

Lack of a priori knowledge The lack of a priori knowledge is two-fold. First, game rules are not provided. The agent objective is to win a game but it has no knowledge about how to score, win/lose or any of the termination conditions. Only the agent's

state observation at current game tick, including the legal actions in the current state, is accessible. However, this state observation does not necessarily cover the whole game state. Secondly, in the corresponding competitions, only some of the game levels are released for training; the agents will be tested on unseen levels.

Real-time constraints Due to the real-time feature of video games, the construction and initialization of a learning agent should not take too long, and during the game playing, an agent needs to decide an action rapidly enough (in the GVGAI competitions, no more than 1s and 40ms, respectively). This ensures a smoother playing experience, similar to when a human player plays a real-time video game. For this reason, in the competitions presented later in Section 3, a learning agent is disqualified in the competition or a *doNothing* action is performed instead of its selection depending on the actual time it takes to select an action.

General game playing Another issue is that an agent trained to perform well on a specific level of a specific game may perform fairly on a similar game level but fails easily on a very different game or level. For instance, a human-level *Super Mario* agent may not survive long in a *PacMan* game; an agent which dominates in *Space Invaders* is probably not able to solve *Sokoban*. Designing a single agent that performs well on a set of different unknown games is not trivial.

Delayed rewards In all the games in the GVGAI framework (as well as in the competitions), the main task is to win the game. However, no winning condition is provided but the instant game score at the current episode. In puzzle and board games, the landscape of instant scores during game-play is usually flat until the last episode, therefore the design of some heuristic and planning ahead are necessary. Defining an appropriate heuristic is not trivial neither for unseen levels of a game or for a set of games.

2 Framework

Different from the planning tracks, no FM is given to the agents in the learning environment, thus, no simulation of games is available. To avoid accessing to the forward model, a new interface was implemented in 2017 on top of the main GVGAI framework. Then, *Philip Bontrager* and *Ruben Rodriguez Torrado* interfaced ittwith OpenAI Gym in 2018. The former one has been used in the first GVGAI Learning Competition organized at the IEEE's 2017 Conference on Computational Intelligence and Games (IEEE CIG 2017), which supports agents written in *Java* and *Python*. Then the second competition was organized at the IEEE's 2018 CIG using a modified

framework interfaced with OpenAI Gym, which only supports agents written in *Python*. The GVGAI framework makes it easy to train and test learning agents potentially on an infinity number of games thanks to the use of VGDL. This is a difficult reinforcement learning problem due to the required generality and limited online decision time.

2.1 GVGAI Learning Framework

A number of games are provided in each game set of which each includes a number of levels (usually five in the framework). Thanks to VGDL and the integrated games provided by the GVGAI framework, this enables the option to design different sets of games and levels.

Execution in a set would have two phases: a *Learning Phase* using N_L levels of the N available and a *Test Phase*, using the other N_T levels with $N_T = N - N_L$. The big picture of these two phases are given below, though there are some differences in details in the 2017's and 2018's competitions.

Learning phase: An agent has a limited amount of time, T_L, for playing a set of M games with N_L learning levels each. It will play each of the N_L levels once, then to be free to choose the next level to play if there is time remaining. The agent is allowed to send the action `abort` to finish the game at any moment, apart from the normal available game actions. The method `result` is called at the end of every game playing regardless if the game has finished normally or been corrupted by the agent using `abort`. Thus, the agent can play as many games as desired, potentially choosing the levels to play in, as long as it respects the time limit T_L.

Test phase: After the learning phase, the trained agent is tested on the N_T test levels that have never been seen during learning. The win rate and statistics of the score and game length over the test trials are used to evaluate the agents.

Note that no FM is accessible during neither of the phases, but learning agents receive an observation of the current game state at every game tick in different form(s), depending on which GVGAI learning environment is used. Both environments are available on-line and can be used for research on general video game learning.

2.2 GVGAI Learning Environment

The GVGAI Learning Environment used in the IEEE CIG 2017's Single-Player Learning Competition is briefly introduced in this section, more technical details about how to set up the framework and the competition procedure are described in the technical manual [12].

The GVGAI Learning Environment is part of the same project as the planning environment[1]. It supports learning agents written in *Java* or *Python*. At every game tick, the agent is allowed to send a valid game action to execute in the game, and, at the same time, to select the format of the next state observation to receive between a serialized JSON game observation (*String*) and a screenshot of the game screen (*PNG*). At any time, the agent can select the format of the game observation to be received at next game tick, using one of the following types:

```
lastSsoType = Types.LEARNING_SSO_TYPE.JSON;   //request for a JSON
lastSsoType = Types.LEARNING_SSO_TYPE.IMAGE;  //request for a screen-shot
lastSsoType = Types.LEARNING_SSO_TYPE.BOTH;   //request for both
```

The choice will be remembered until the agent makes another choice using the above commands. An example of the screenshot is given in Figure 1.

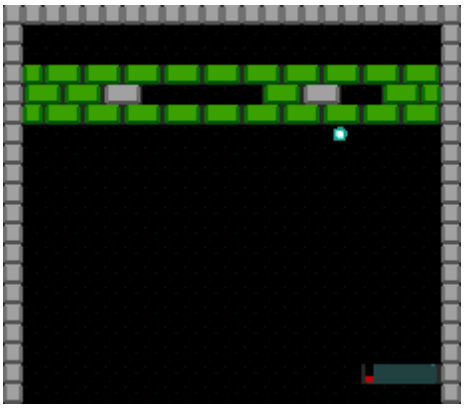

Fig. 1: Example: screenshot of a game screen.

Below is an example of the serialized state observation.

```
SerializableStateObservation{
    phase=ACT, isValidation=false, gameScore=8.0,
    gameTick=150, gameWinner=NO_WINNER, isGameOver=false,
    worldDimension=[230.0, 200.0], blockSize=10, noOfPlayers=1, ...
    ...
    availableActions=[ACTION_USE, ACTION_LEFT, ACTION_RIGHT],
    avatarResources={},
```

[1] https://github.com/GAIGResearch/GVGAI

```
8       observationGrid={
9           Observation{category=6, itype=0, obsID=578, position=0.0 : 0.0,
10          reference=-1.0 : -1.0, sqDist=2.0}
11          ...
12          Observation{category=6, itype=11, obsID=733, position=150.0 : 40.0,
                reference=-1.0 : -1.0, sqDist=24482.0}
13      },
14      resourcesPositions=null, portalsPositions=null,
            fromAvatarSpritesPositions=null}
15  }
```

As in the planning environment, a *Java* or *Python* agent should inherit from an abstract class `AbstractPlayer`, implement the constructor and three methods: `act`, `init` and `result`. The class must be named `Agent.java` or `Agent.py`.

Implementation of a learning agent

`Agent()` The constructor is called once per game and must finish in no more than `START_TIME` (by default $1s$) of CPU time, thus it is called once during the learning and test phases of each game.

`init(SerializableStateObservation sso, ElapsedCpuTimer elapsedTimer)` After creating an agent, `init()` is called before every single game run. It should finish in no more than `INITIALIZATION_TIME` (by default $1s$) of CPU time.

`act(SerializableStateObservation sso, ElapsedCpuTimer elapsedTimer)` At each game tick, `act()` is called and determines the next action of the agent within the prescribed CPU time `ACTION_TIME` (by default $40ms$). The possible actions are `ACTION_LEFT, ACTION_RIGHT, ACTION_UP, ACTION_DOWN, ACTION_USE` and `ACTION_NIL` (do nothing). The agent will be disqualified immediately if more than `ACTION_TIME_DISQ` (by default $50ms$) is taken. Otherwise, a `NIL` action (do nothing) is applied. Note that it is possible that in a game or at a game state, not all the actions listed above are available (legal) actions, but `ACTION_NIL` always is.

`result(SerializableStateObservation sso, ElapsedCpuTimer elapsedTimer)` This method is called at the end of every game. It has no time limit, so the agent doesn't get penalized for overspending other than the `TOTAL_LEARNING_TIME` indicated. The agent can play with the time it spends on the `result` call to do more learning or to play more games. At each call of `result`, an action or a level number should be returned.

Termination A game playing terminates when the player wins/loses the game or the maximal game ticks (`MAX_TIMESTEPS`) is reached.

Time out If the agent returns an action after `ACTION_TIME` but no more than `ACTION_TIME_DISQ`, then the action `ACTION_NIL` will be performed.

Disqualification If the agent returns an action after `ACTION_TIME_DISQ`, the agent is disqualified and loses the game.

Parameters The notation and corresponding parameters in the framework are summarized in Table 1, as well as the default values.

Parameters for client (agent)		
Variable	**Default value**	**Usage**
START_TIME	1s	Time for agent's constructor
INITIALIZATION_TIME	1s	Time for `init()`
ACTION_TIME	40ms	Time for returning an action per tick
ACTION_TIME_DISQ	50ms	Threshold for disqualification per tick
TOTAL_LEARNING_TIME	5min	Time allowed for learning a game
EXTRA_LEARNING_TIME	1s	Extra learning time
SOCKET_PORT	8080	Socket port for communication
Parameters for server		
Variable	**Default**	**Usage**
MAX_TIMESTEPS	1000	Maximal game ticks a game can run
VALIDATION_TIMES	10	Number of episodes for validation

Table 1: The main parameters in the learning framework.

2.3 GVGAI Gym Environment

GVGAI Gym is a result of interfacing the GVGAI framework to the OpenAI Gym environment by Ruben Rodriguez Torrado and Philip Bontrager, PhD candidates at the New York University School of Engineering [15]. Beside the more user-friendly interface, a learning agent still receives a screenshot of the current game screen and game score, then returns a valid action at every game tick. An example of a random agent that plays first level of *Aliens* using GVGAI Gym is illustrated in Figure 2. We compare the GVGAI Gym implementation and the original GVGAI environment in Table 2.

	GVGAI Planning 1-Player	GVGAI Planning 2-Player	GVGAI Learning	GVGAI Gym
Similarities	• Play unseen games, no game rules available • Access to game score, tick, if terminated • Access to legal actions • Access to observation of current game state			
Forward model?	Yes	No		NO
History events?	Yes	No		NO
State Observation?	*Java* object	*String* or *PNG*		*PNG*
	Java	Java &Python		Python

Table 2: Comparison of the planning and learning environments.

```python
import gym
import gym_gvgai

env = gym.make('gvgai-aliens-lvl0-v0')
env.reset()

score = 0
for i in range(2000):
    action_id = env.action_space.sample()
    state, reward, isOver, info = env.step(action_id)
    score += reward
    print("Action " + str(action_id) + " played at game tick " + str(i+1) + ", reward=" + str(reward) + ", new score=" + str(score))
    if isOver:
        print("Game over at game tick " + str(i+1) + " with player " + info['winner'])
        break
```

Fig. 2: Sample code of randomly playing the first level of *Aliens* using GVGAI Gym.

2.4 Comparing to Other Learning Frameworks

Other general frameworks like OpenAI Gym [5], Arcade Learning Environment (ALE) [3] or Microsoft Malmö [7] contain a great number of single-/multi-player, model-free or model-based tasks. Interfacing with these systems would greatly increase the number of available games which all GVGAI agents could play via a common API. This would also open the framework to 3D games, an important section of the environments the current benchmark does not cover.

At the time of writing, ALE [3] offers higher-quality games than GVGAI as they were home-console commercial games of a few decades ago, whereas the GVGAI provides a structured API (information available via *Java* objects, or *JSON* interface, or screen capture); the agents are tested on unseen games; and there is potentially infinite supply of games. In GVGAI terms, ALE offers just two tracks: single-player learning and planning, with the learning track being the more widely used. The GVGAI framework has the potential to be expanded by adding a two-player learning track, which will offer more open-ended challenges. This is outside of the current scope of ALE. Again, thanks to VGDL, it is much more easier to create new games or to create new levels for these games, using the GVGAI Learning environment or

GVGAI Gym. It is also easy to automatically generate variations on existing VGDL games and their levels. Thus, the users can apply procedural content generation to generate game and level variations for training and testing their learning agents. GVGAI is more easily extensible than ALE, and offers a solution to overfitting.

3 GVGAI Learning Competitions

At the time that this book is written, only two GVGAI learning competitions have been organized. In this section, we describe the competition rules and core challenges of individual competition besides the common challenges of both that have been presented in Section 2.1.

3.1 Competition using the GVGAI Learning Environment

The first GVGAI learning competition was organized at the IEEE's 2017 Conference on Computational Intelligence and Games (IEEE CIG 2017).

Competition Procedure and Rules In this competition, 10 games are used, of which 3 levels are given for training and 2 private levels are used for testing. The set of 10 games used in this learning competition is the training set 1 of the 2017 GVGAI Single-Player Planning Competition.

The *Learning Phase* consists of two steps, referred to as *Learning Phase 1* and *Learning Phase 2*. The whole procedure is illustrated in Algorithm 1 and Figure 3. An agent has a limited duration 5 *mins* (legal learning duration) in total for both training phases. The communication time is not included by the `Timer`. In case that 5min has been used up, the results and observation of the game will still be sent to the agent and the agent will have no more than 1 *second* before the test.

Fig. 3: Learning and test phases for one game in the 2017 single-player learning competition. In the competition, an agent will be executed on a set of (usually 10) unknown games, thus this process will repeat 10 times.

During the *Learning phase 1* of each game (lines 2-5 of Algorithm 1), an agent plays once the three training levels sequentially. At the end of each level, whether the game has terminated normally or the agent forces to terminate the game (using `abort`), the server will send the results of the (possibly unfinished) game to the agent before termination.

Algorithm 1 Main procedure of the 2017 learning competition.

Require: $\mathcal{G}$ set of games
Require: $\mathcal{L}$ set of training levels (per game)
Require: $\mathcal{T}$ set of training levels (per game)
Require: π and agent

```
1: for each game G ∈ 𝒢 do
2:     for each game level ∈ G_ℒ do
3:         Let π play level once
4:         nextLevel ← π.result()
5:     while Time is not elapsed do
6:         Let π play nextLevel once
7:         nextLevel ← π.result()
8: for each game G ∈ 𝒢 do
9:     for each game level ∈ G_𝒯 do
10:        RES_G ← Results of 10 games of π on level
11: return RES_𝒢
```

After having finished *Learning phase 1*, the agent is free to select the next level to play (from the three training levels) by calling the method `int result()` (detailed in Section 2.2). If the returned index of the selected level is not a valid index, then a random index from the valid indexes will be passed and a new game will start. This step (*Learning Phase 2*) (lines 6-9 of Algorithm 1) is repeated until the legal learning time has expired.

After looping over the whole set of games, the *Test Phase* (lines 11-15 of Algorithm 1) starts. The trained agent repeatedly plays 10 times the private test levels sequentially. There is no more total time limit, but the agent still needs to respect the time limits for the methods `init`, `act` and `result`, and can continue learning during the game playing.

Challenges of the Competition Beside the challenges of learning in GVGAI (Section 2.1), there are some other crucial problems due to the competition rules, such as how to select which level to train next; how to distribute the total learning duration; which type of state observation to receive; training different models for different games or training one unique model; etc. None of these is trivial to decide and no entry performed reasonably in the competition. Therefore, the competition rules were changed for 2018.

Competition results In the 2017 edition of GVGAI learning competition, the execution of controllers was divided into two phases: learning and validation. In the

learning phase, each controller has a limited amount of time, 5 min, for learning the first three levels of each game. The agent could play as many times as desired, choosing among these three levels, as long as the 5 min time limit is respected. In the validation phase, the controller plays 10 times the levels 4 and 5 sequentially. The results obtained in these validation levels are the ones used in the competition to rank the entries. Besides the two sample random agents written in Java and Python and one sample agent using Sarsa written in Java, the first GVGAI single-player learning track received three submissions written in Java and one in Python [11]. The results are illustrated in Table 3. The winner of this track is a naive implementation of the Q-Learning algorithm (Section 4.5).

Agent	**Training set**		**Test set**	
	Score	**Ranking**	**Score**	**Ranking**
kkunan	125	6	**184**	**1**
sampleRandom†	**154**	**2**	**178**	**2**
DontUnderestimateUchiha	149	3	158	3
sampleLearner†	149	4	152	4
ercumentilhan	**179**	**1**	134	5
YOLOBOT	132	5	112	6

Table 3: Score and ranking of the submitted agents in the 2017's GVGAI Learning Competition. †denotes a sample controller.

Table 4 compares the best scores by single-player planning and learning agents on the same test set. Note that one of the games in the test set is removed from the final ranking due to bugs found in the game itself. The best performance of learning agents on tested games is far worse than what the planning agents can achieve.

3.2 Competition using the GVGAI Gym

The GVGAI Gym has been used in the learning competition organized at the IEEE's 2018 Conference on Computational Intelligence and Games (IEEE CIG 2018).

Competition Procedure and Rules The competition rules have been changed based on the experience of the first GVGAI learning competition. The main procedures are as follows.

1. Three games with two public levels each were given for training one month before the submission deadline. The participants were free to train their agents privately with no constraints and can use any computational resource that they had.

Game	1-P Planning	1-P Learning	
	Best score	Best score	Agent
G2	109.00 ± 38.19	31.5 ± 14.65	sampleRandom†
G3	1.00 ± 0.00	0±0	*
G4	1.00 ± 0.00	0.2 ± 0.09	kkunan
G5	216.00 ± 24.00	1±0	*
G6	5.60 ± 0.78	3.45± 0.44	DontUnderestimateUchiha
G7	31696.10 ± 6975.78	29371.95±2296.91	kkunan
G8	1116.90 ± 660.84	35.15±8.48	kkunan
G9	1.00 ± 0.00	0.05± 0.05	sampleRandom†
G10	56.70 ± 25.23	2.75 ± 2.04	sampleLearner†

Table 4: Table compares the best scores by single-player planning and learning agents on the same test set. Note that one of the games in the test set is removed from the final ranking due to bugs in the game itself. †denotes a sample controller.

2. After the submission deadline, three new private levels of each of the three games are used for validation. The validation phase was ran by the organizers using one single laptop. Each experiment has been repeated 10 times. During validation, each agent has $100ms$ per game tick to select an action.

Three test games with very different aspects have been designed for this competition. The game 1 is modified from *Aliens*, and the only difference is that the avatar is allowed to shoot and move in four directions instead of one and two directions, respectively. The game 2 is a puzzle game called *Lights On*, in which the avatar wins if it turns on the lights shown on the right end of the game screen (e.g., Figure 4). The game 3 is modified from a deceptive game by [1], called *DeceptiCoins*.

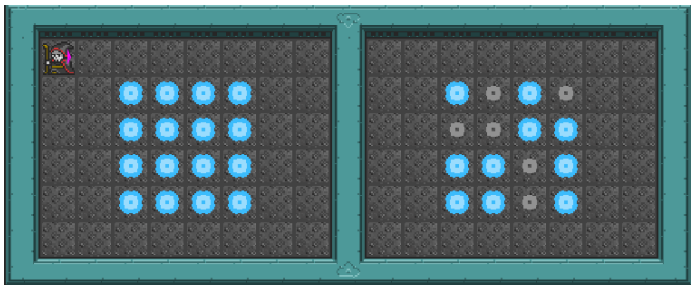

Fig. 4: Screenshot of the puzzle game *Lights On* that has been used in the second GVGAI learning competition. The screenshot illustrates the initial state of a level.

Table 5: Score and ranking of the submitted agents in the 2018's GVGAI Learning Competition. †denotes a sample controller.

Game	**Game 1**			**Game 2**			**Game 3**			**Ranking**
Level	**3**	**4**	**5**	**3**	**4**	**5**	**3**	**4**	**5**	
fraBot-RL-Sarsa	-2	**1**	-1	0	0	0	**2**	**3**	**2**	1
fraBot-RL-QLearning	-2	-1	-2	0	0	0	1	0	**2**	2
Random††	-0.5	**0.2**	**-0.1**	0	0	0	**3.5**	**0.7**	**2.7**	3
DQN†	**61.5**	-1	**0.3**	0	0	0	-	-	-	-
Prioritized Dueling DQN†	**36.8**	-1	-2	0	0	0	-	-	-	-
A2C†	8.1	-1	-2	0	0	0	-	-	-	-
OLETS Planning Agent	41.7	48.6	3.1	0	0	2.2	4.2	8.1	14	-

Challenges of the Competition The new competition rules and more user-friendly framework make the users focus more on the learning algorithms, including those that are already compatible with Open AI Gym. The fact that users are free to train their agents privately using as much as learning time and computational resources as they like enables more opportunities and possibilities. However, some classic learning algorithms are not able to handle the different dimensions of the screen that distinct games have.

Competition results This edition of the competition received only 2 entries, *fraBot-RL-QLearning* and *fraBot-RL-Sarsa*, submitted by the same group of contributors from the Frankfurt University of Applied Science. The results of the entries and sample agents (*random*, *DQN*, *Prioritized Dueling DQN* and *A2C* [15]) are summarized in Table 5. For comparison, the planning agent *OLETS* (with access to the forward model) is included. *DQN* and *Prioritized Dueling DQN* are outstanding on level 3 (test level) of the game 1, because the level 3 is very similar to the level 2 (training level). Interestingly, the sample learning agent *DQN* outperformed *OLETS* on the third level of game 1. For instance, the baseline agents *DQN*, *Prioritized Dueling DQN* and *A2C* are not able to learn the game *DeceptiCoins* due to the different game screen dimensions in different levels, despite their outstanding performance on the test levels of the game modified from *Aliens*.

4 Competition entries

This section describes first the approaches that tackled the challenge set in the single-player learning track of the 2017 and 2018 competitions, and then moves to other approaches.

4.1 Random agent (Sample agents)

A sample random agent, which selects an action uniformly at random at every game tick, is included in the framework (in both Java and Python) for the purposes of testing. This agent is also meant to be taken as a baseline: a learner is expected to perform better than an agent which acts randomly and does not undertake any learning.

4.2 DRL algorithms (Sample agents)

Using the new GVGAI Gym, Torrado et al. [15] compared three implemented Deep Reinforcement Learning algorithms of the OpenAI Gym, Deep Q-Network (DQN), Prioritized Dueling DQN and Advance Actor-Critic (A2C), on eight GVGAI games with various difficulties and game rules. All the three RL agents perform well on most of the games, however, DQNs and A2C perform badly when no game score is given during a game playing (only win or loss is given when a game terminates). These three agents have been used as sample agents in the learning competition organized at IEEE CIG 2018.

4.3 Multi-armed bandit algorithm

DontUnderestimateUchiha by K. Kunanusont is based on two popular Multi-Armed Bandit (MAB) algorithms, ϵ-Decreasing Greedy Algorithm and Upper Confidence Bounds (UCB). At any game tick T, the current *best* action with probability $1 - \epsilon_T$ is picked, otherwise an action is uniformly randomly selected. The *best* action at time T is determined using UCB with increment of score as reward. This is a very interesting combination, as the UCB-style selection and the ϵ-Decreasing Greedy Algorithm both aim at balancing the trade-off between exploiting more the best-so-far action and exploring others. Additionally, ϵ_0 is set to 0.5 and it decreases slowly along time, formalized as $\epsilon_T = \epsilon_0 - 0.0001T$. According to the competition setting, all games will last longer than $2,000$ game ticks, so $\forall T \in \{1, \ldots, 2000\}$, $0.5 \geq \epsilon_T \geq 0.3$. As a result, random decisions are made for approximately 40% time.

4.4 Sarsa

sampleLearner, *ercumentilhan* and *fraBot-RL-Sarsa* are based on the State-Action-Reward-State-Action (Sarsa) algorithm [13]. The *sampleLearner* and *ercumentilhan* use a subset of the whole game state information to build a new state to reduce the amount of information to be saved and to take into account similar situations. The main difference is that the former uses a square region with fixed size centered at

the avatar's position, while the latter uses a first-person view with a fixed distance. *fraBot-RL-Sarsa* uses Sarsa, and it uses the entire screenshot of the game screen as input provided by GVGAI Gym. The agent has been trained using 1000 episodes for each level of each game, and the total training time was 48 hours.

4.5 Q-learning

kkunan, by K. Kunanusont, is a simple Q-learning [13] agent using most of the avatar's current information as features, which a few exceptions (such as avatar's health and screen size, as these elements that vary greatly from game to game). The reward at game tick $t + 1$ is defined as the difference between the score at $t + 1$ and the one at t. The learning rate α and discounted factor γ are manually set to 0.05 and 0.8. During the *learning phase*, a random action is performed with probability $\epsilon = 0.1$, otherwise, the best action is selected. During the *validation phase*, the best action is always selected. Despite it's simplicity, it won the the first track in 2017. *fraBot-RL-QLearning* uses the Q-Learning algorithm. It has been trained using 1000 episodes for each level of each game, and the total training time was 48 hours.

4.6 Tree search methods

YOLOBOT is an adaption of the *YOLOBOT* planning agent (as described previously in Chapter 4). As the FM is no more accessible in the learning track, the MCTS is substituted by a greedy algorithm to pick the action that minimizes the distance to the chosen object at most. According to the authors, the poor performance of *YOLOBOT* in the learning track, contrary to its success in the planning tracks, was due to the collision model created by themselves that did not work well.

4.7 Other learning agents

One of the first works that used this framework as a learning environment was carried out by Samothrakis et al. [14], who employed Neuro-Evolution in 10 games of the benchmark. Concretely, the authors experimented with Separable Natural Evolution Strategies (S-NES) using two different policies (ϵ-greedy versus Softmax) and a linear function approximator versus a neural network as a state evaluation function. Features like score, game status, avatar and other sprites information were used to evolve learners during 1000 episodes. Results show that ϵ-greedy with a linear function approximator was the better combination to learn how to maximize scores on each game.

Braylan and Miikkulainen [4] performed a study in which the objective was to learn a forward model on 30 games. The objective was to learn the next state from the

current one plus an action, where the state is defined as a collection of attribute values of the sprites (spawns, directions, movements, etc.), by means of logistic regression. Additionally, the authors transfer the learnt object models from game to game, under the assumption that many mechanics and behaviours are transferable between them. Experiments showed the effective value of object model transfer in the accuracy of learning forward models, resulting in these agents being stronger at exploration.

Also in a learning setting, Kunanusont et al. [9] [10] developed agents that were able to play several games via screen capture. In particular, the authors employed a Deep Q-Network in seven games of the framework of increasing complexity, and included several enhancements to GVGAI to deal with different screen sizes and a non-visualization game mode. Results showed that the approach allowed the agent to learn how to play in both deterministic and stochastic games, achieving a higher winning rate and game score as the number of episodes increased.

Apeldoorn and Kern-Isberner [2] proposed a learning agent which rapidly determines and exploits heuristics in an unknown environment by using a hybrid symbolic/sub-symbolic agent model. The proposed agent-based model learned the weighted state-action pairs using a sub-symbolic learning approach. The proposed agent has been tested on a single-player stochastic game, *Camel Race*, from the GVGAI framework, and won more than half of the games in different levels within the first 100 game ticks, while the standard Q-Learning agent never won given the same game length. Based on [2], Dockhorn and Apeldoorn [6] used exception-tolerant Hierarchical Knowledge Bases (HKBs) to learn the approximated forward model and tested the approach on the 2017 GVGAI Learning track framework, respecting the competition rules. The proposed agent beats the best entry in the learning competition organized at CIG 2017 [6], but still performed far worse than the best planning agents, which have access to the real forward models.

Finally, Justesen et al. [8] implemented A2C within the GVGAI-Gym interface in a training environment that allows learning by procedurally generating new levels. By varying the levels in which the agent plays, the resulting learning is more general and does not overfit to specific levels. The level generator creates levels at each episode, producing them in a slowly increasing level of difficulty in response to the observed agent performance.

4.8 Discussion

The presented agents differ between each other in the input game state (JSON string or screen capture), the amount of learning time, the algorithm used. Additionally, some of the agents have been tested on a different set of games and sometimes using different game length (i.e., maximal number of game ticks allowed). None of the

agents, which were submitted to the 2017 learning competition, using the classic GVGAI framework, have used screen capture.

The Sarsa-based agents performed surprisingly bad in the competition, probably due to the arbitrarily chosen parameters and very short learning time. Also, learning three levels and testing on two more difficult levels given only 5 min learning time is a difficult task. An agent should take care of the learning budget distribution and decide when to stop learning a level and to proceed the next one.

The learning agent using exception-tolerant HKBs [6] learns fast. However, when longer learning time is allowed, it is dominated by Deep Reinforcement Learning (DRL) agents. Out of the eight games tested by Torrado et al. [15], none of the tested three DRL algorithms outperformed the planning agents on six games. However, on the heavily stochastic game Seaquest, A2C achieved almost double score than the best planning agent, MCTS.

5 Summary

In this chapter, we present two platforms for the GVGAI learning challenges, which can be used for testing reinforcement learning algorithms, as well as some baseline agents. This chapter also reviews the 2017 and 2018 GVGAI learning competitions organised using each of the platforms. Thanks to the use of VGDL, the platforms have the potential of designing new games by humans and AIs for training reinforcement learning agents. In particular, the GVGAI Gym is easy to use to implement and compare agents. We believe this platform can be used in multiple research directions, including designing reinforcement learning agents for a specific task, investigating artificial general intelligence, and evaluating how different algorithms can learn and evolve to understand various changing environments.

References

1. D. Anderson, M. Stephenson, J. Togelius, C. Salge, J. Levine, and J. Renz, "Deceptive Games," *arXiv preprint arXiv:1802.00048*, 2018.
2. D. Apeldoorn and G. Kern-Isberner, "An Agent-Based Learning Approach for Finding and Exploiting Heuristics in Unknown Environments," in *COMMONSENSE*, 2017.
3. M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: an evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, no. 1, pp. 253–279, 2013.
4. A. Braylan and R. Miikkulainen, "Object-Model Transfer in the General Video Game Domain," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.
5. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai Gym," *arXiv preprint arXiv:1606.01540*, 2016.

6. A. Dockhorn and D. Apeldoorn, "Forward Model Approximation for General Video Game Learning," in *Computational Intelligence and Games (CIG), IEEE Conference on*, 2018.
7. M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, "The Malmo Platform for Artificial Intelligence Experimentation," in *IJCAI*, 2016, pp. 4246–4247.
8. N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation," *arXiv:1806.10729*, 2018.
9. K. Kunanusont, "General Video Game Artificial Intelligence: Learning from Screen Capture," Master's thesis, University of Essex, 2016.
10. K. Kunanusont, S. M. Lucas, and D. Pérez-Liébana, "General Video Game AI: Learning from Screen Capture," in *2017 IEEE Conference on Evolutionary Computation (CEC)*. IEEE, 2017.
11. J. Liu, "GVGAI Single-Player Learning Competition at IEEE CIG17," 2017. [Online]. Available: https://www.slideshare.net/ljialin126/gvgai-singleplayer-learning-competition-at-ieee-cig17
12. J. Liu, D. Perez-Liebana, and S. M. Lucas, "The Single-Player GVGAI Learning Framework - Technical Manual," 2017. [Online]. Available: http://www.liujialin.tech/publications/GVGAISingleLearning_manual.pdf
13. S. J. Russell and P. Norvig, *Artificial Intelligence: a Modern Approach*. Malaysia; Pearson Education Limited,, 2016.
14. S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for General Video Game Playing," in *Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 200–207.
15. R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep Reinforcement Learning in the General Video Game AI framework," in *IEEE Conference on Computational Intelligence and Games (CIG)*, 2018.