Gym Hero: A Research Environment for Reinforcement Learning Agents in Rhythm Games

Rômulo Freire Férrer Filho
Teleinformatics Engineering Department (DETI)
Federal University of Ceará (UFC)
Fortaleza, Brazil
romulofffufc@gmail.com

Yuri Lenon Barbosa Nogueira
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
yuri@dc.ufc.br

Creto Augusto Vidal
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
cvidal@dc.ufc.br

Joaquim Bento Cavalcante-Neto
Department of Computing (DC)
Federal University of Ceará (UFC)
Fortaleza, Brazil
joaquimb@dc.ufc.br

Paulo Bruno de Sousa Serafim
Instituto Atlântico
Fortaleza, Brazil
paulo_serafim@atlantico.com.br

***Abstract*—This work presents a Reinforcement Learning environment, called Gym Hero, based on the game Guitar Hero. It consists of a similar game implementation, developed using the graphics engine PyGame, with four difficulty levels, and able to randomly generate tracks. On top of the game, we implemented a Gym environment to train and evaluate Reinforcement Learning agents. In order to assess the environment's capacity as a suitable learning tool, we ran a set of experiments to train three autonomous agents using Deep Reinforcement Learning. Each agent was trained on a different level using Deep Q-Networks, a technique that combines Reinforcement Learning with Deep Neural Networks. The input of the network is only the pixels of the screen. We show that the agents were capable of learning the expected behaviors to play the game. The obtained results validate the proposed environment as capable of evaluating autonomous agents on Reinforcement Learning tasks.**

***Keywords*—autonomous agents, reinforcement learning, deep learning, reinforcement learning environments, rhythm games, guitar hero**

I. Introduction

The interest in building machines capable of beating humans in games is very old. In the eighteenth century, a machine known as "The Turk" won chess matches against famous opponents, such as Napoleon Bonaparte and Benjamin Franklin. However, in the next century, that feat was revealed to be a scam. Inside the machine, there was an experienced chess player that decided which moves should be performed at each moment [1]. At the end of the twentieth century, in 1997, the victory of the IBM's supercomputer, Deep Blue, over the world's chess champion, Garry Kasparov, reinforced that interest [2].

Nowadays, works combining autonomous agents with videogames obtained interesting results, such as the victory of an agent over the world's champion of Go [3] and the first victory of an intelligent agent over a professional StarCraft II[1] player [4]. The aforementioned advances were made possible by the creation of a new machine learning approach called Deep Reinforcement Learning (DRL). Mnih *et al.* [5], [6] combined Deep Learning models [7], which stand out in image-related problems, with the Reinforcement Learning (RL) paradigm [8] to develop an agent capable of beating human opponents in several games from the Atari 2600[2] console, using only the game screen as input.

However, rhythm games, a subgenre of action games that use music as the central part of their mechanics, were little explored in RL works, despite their great popularity [9]. Franchises such as Dance Dance Revolution[3], Guitar Hero[4] and Just Dance[5] bring new challenges to intelligent agents, because they require coordination of fast and accurate movements following a characteristic rhythm pattern.

In this work, we present Gym Hero[6], a new environment focused on rhythm games for training and evaluation of Reinforcement Learning algorithms. We implemented a rhythm game inspired by Guitar Hero, which serves as the basis for the learning environment. Then, we implemented a Gym environment [10] on top of the game, which is used to train and evaluate Deep Reinforcement Learning agents. We trained a set of three autonomous agents using Deep Q-Networks [5], [6] and show that they can learn to play Gym Hero. The interaction dynamics between Gym Hero and an agent is illustrated in Fig. 1.

This paper is organized as follows. In Section II, we discuss similar works, presenting their main features. In Section III, we present the theoretical basis of DRL and a brief description of rhythm games. In Section IV, the game, environment, and agent development are described in detail. Then, in Section

[1]StarCraft is a trademark of Blizzard Entertainment, Inc.

[2]Atari 2600 is a trademark of Atari SA.

[3]Dance Dance Revolution is a trademark of Konami Digital Entertainment Co., Ltd.

[4]Guitar Hero is a trademark of Activision Publishing, Inc.

[5]Just Dance is a trademark of Ubisoft Entertainment SA.

[6]Code available at: https://github.com/romulofff/gym-hero

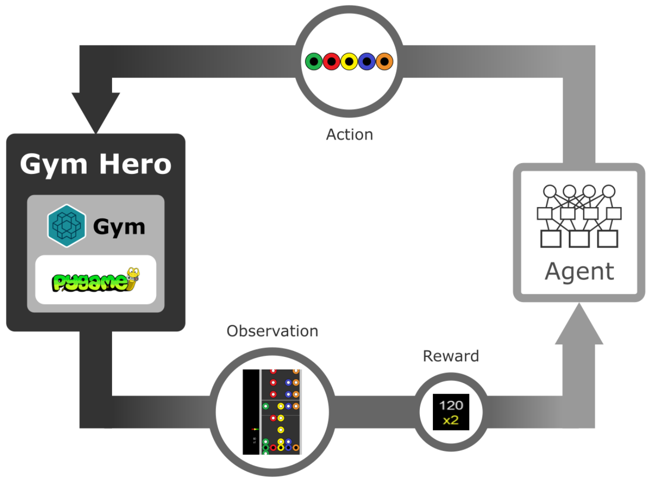


Fig. 1. Gym Hero was written using PyGame and provides a Gym interface. An external agent would select one of the possible actions. Then, Gym Hero gives the corresponding reward to the executed action and presents a new observation to the agent.

V, the results obtained by training an autonomous agent using the proposed environment are presented and discussed. Finally, in Section VI, we present the final considerations, as well as suggestions for some future works.

II. Related Work

In this section, we describe works that report attempts to play Guitar Hero using Computer Vision and robots that use videogame controllers and electronic actuators to play the notes at the right time; and, then, we discuss about an environment for Reinforcement Learning research.

Grybot [11] is a robotic system implemented using a Field Programmable Gate Array (FPGA), and connected to an electromechanical system, which is responsible for pressing the buttons of a real controller. It analyzes a set of predefined pixels that correspond to the game's Region of Interest (ROI), and, after checking the notes' colors and positions, sends a signal for the actuator to press the button. The authors reported an average accuracy of 97% on all difficulty levels.

CythBot is a non-humanoid robot, which was designed to win battles against human players on Guitar Hero [12]. It differs from Grybot because it does not use colors to detect the notes. Instead, the authors chose to track the pixel intensity in a specific ROI. If that value is higher than a threshold, a signal is sent to the actuator so that it can press the button after a delay. The authors reported an average accuracy of 80%.

GuitarHeroNoid [13] is another robot that was developed to play Guitar Hero songs. It is divided into two subsystems: a brain and a body. The brain receives the game's image from the console using a capture card; converts that image to the HSV (hue, saturation, and value) color space; and applies a threshold filter in order to obtain a binary image where the notes stand out. Also, a ROI is defined, and, if the sum of the pixel values inside the ROI reaches a threshold, this set of pixels is considered a note. Then, 250 milliseconds after the note is detected, a signal is sent to the body so that it plays the correct combination of buttons.

The *Rockband Robot* [14] plays Rock Band instead of Guitar Hero. It follows the same idea from the previous robots, using a set of solenoid actuators and an image acquisition system to play the game. The main difference is that it uses the HSL (hue, saturation, and luminosity) color space, from which the saturation value is used for detection. The robot was tested against human players and the authors reported wins in 90% of the matches.

Concerning Reinforcement Learning Environments used for that kind of research, Brockman *et al.* [10] presented Gym, a set of tools that allows one to build, customize and distribute scenarios for Reinforcement Learning research. Among the many features of this set of tools, the focus on the environment, rather than on the agent, stands out. That allows multiple agents to be implemented using a different technique for each agent, which simplifies the comparison between them. Due to Gym's easy customization, various environments can be adapted to follow Gym's standard. Moreover, the Gym provides a Python Application Programming Interface (API), contributing to its popularization and use in several works.

III. Background

A. Guitar Hero

A rhythm game is a subgenre of action games that challenges the player to follow a rhythm [15] while pressing buttons on a controller or performing movements in front of motion detection devices. This genre is divided into two main categories: dance games, such as Just Dance and Dance Dance Revolution; and music games, like Guitar Hero and Rock Band. Among the music-focused games, the Guitar Hero franchise stands out as one of the most successful games [9].

The game mechanics of Guitar Hero uses a guitar-shaped controller, which has five colored buttons that match buttons on the screen. The player needs to hold down the correct buttons and press the strum bar when the notes are over the button indicators on the screen. Originally, the game used custom *midi* [16] files to encode the songs. However, with the popularization of Guitar Hero clones, such as Clone Hero, the *chart* file format was created and adopted by the developers [17], [18].

The *chart* format consists of a text file divided into five sections: the Song, SyncTrack, and Events sections; plus two song difficulty sections – ExpertSingle and EasySingle. The Song section contains the music metadata, the name, the offset, and the resolution which represents the number of ticks on a beat. A tick is an imaginary time unit within a song. Other metadata that may be in the *chart* are Difficulty, Genre, and Audio files' names of each instrument. The SyncTrack section describes changes in the song's number of beats per minute (BPM). Each line in this section is in the format $0 = B$ 120000. The first number is the tick where this change occurs, the "B" represents a BPM change and the number 120000 is the new BPM value. In the example, the BPM should be changed to 120000 on tick 0. The Events section describes animations and lightning effects, but it is only used when the song has *midi* files.

The song's difficulty sections define: when each note must be played, the note type, the color of the note, and its duration. The lines appear in a similar fashion to those in the Events section. The first number indicates in which tick the note should be played. The letter represents the type of the note: "N" means a standard note, and "S" a star note, which fills a special power gauge, that when activated doubles the received score. After that, there is a number from 0 to 5. The numbers from 0 to 4 indicate the colors green, red, yellow, blue and orange respectively. The number 5 indicates that the note above is a special note. The last value represents the note's duration. These are a few examples of notes in a *chart* file:

3360 = N 2 0 $\rightarrow$ At tick 3360, there is a yellow note of standard duration.

7420 = N 3 300 $\rightarrow$ At tick 7420, there is a blue note of duration 300 ticks.

28560 = N 0 0 $\rightarrow$ At tick 28560, there is a green note of standard duration.

28560 = N 5 0 $\rightarrow$ At tick 28560 the green note is a special note.

B. Reinforcement Learning

Reinforcement Learning is a machine learning paradigm focused on solving sequential decision problems through interaction between an agent and an environment. The agent performs actions in the environment and receives rewards as a result. Ultimately, the goal of the agent is to maximize the total sum of those rewards [8]. At time t, the agent is in state S_t and executes an action A_t. Then, the environment returns a state S_{t+1} and a reward R_{t+1}. To maximize the sum of rewards, the agent must find the best action to execute at every state. That action is the one associated with the greatest sum of discounted rewards,

$$R_t = \sum_{k=0}^{T} \gamma^k \; r_{t+k+1}, \tag{1}$$

where T is the last iteration, k is the current iteration and γ is the discount factor. Using γ, we can guarantee that the sum converges even when $T = \infty$. Intuitively, we can say that the immediate rewards receive a bigger priority than the future ones. The mapping of each state to an action is called policy (π).

C. Q-Learning

The goal of Q-Learning [19], [20] is to generate the optimal policy π^*. To achieve that, the algorithm updates a scalar value $Q(s, a)$, for every state-action pair, according to the update rule

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[y - Q(S_t, A_t) \right], \tag{2}$$

where α is the learning rate, $0 \leq \alpha \leq 1$, and

$$y = R_{t+1} + \gamma \max_a Q(S_{t+1}, a). \tag{3}$$

Throughout the learning process, the approximation will converge to the optimal values $Q_*(s, a)$. In simple tasks, the state-action pairs can be represented by a bidimensional array, in which the rows are states and the columns are actions [8]. However, in more complex tasks, it is necessary to use more robust representations.

D. Deep Q-Networks

When the number of states in the environment is too large, using a table to store the Q values becomes computationally unfeasible. Therefore, it is common to use an approximation function $Q(s, a; \theta)$, where the array θ contains the parameters of the estimator, such that $Q(s, a; \theta) \approx Q_*(s, a)$. When that approximation is a Deep Neural Network (DNN), the combination of Q-Learning with DNNs is called Deep Q-Networks [5], [6]. To train the neural network, the update rule from Q-Learning must be transformed into a loss function

$$L_i(\theta_i) = \mathbb{E}\left[(y_i - Q(s, a; \theta_i))^2 \right], \tag{4}$$

with

$$y_t = r_t + \gamma \cdot max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}). \tag{5}$$

Therefore, a trained neural network using the loss $L_i(\theta_i)$ will be able to approximate $Q_*(s, a)$.

E. DRL Environments

The first Deep Reinforcement Learning projects utilized games from the Atari 2600 console as testing environments. Since then, several environments have been introduced, like ViZDoom [21], OpenAI Gym [10], and Unity ML-Agents [22]. For an environment to be useful for Reinforcement Learning tasks, it must allow the agent to interact with the game, to perform actions in the environment, to receive information in the form of states or observations, and to be rewarded based on its actions [23]. It is in this context that rhythm games, such as Guitar Hero, appear as possible training environment for Reinforcement Learning algorithms, and challenge the agent's perception of rhythm.

IV. Methodology

Gym Hero, the Reinforcement Learning environment presented in this paper, was based on the game Guitar Hero. Although similar games do exist, the control of the inner dynamics of the game makes ours unique, with a design that meets the requirements of a learning environment [23]:

- it allows the agent to interact with the game, by executing actions within it;
- it returns information about the game state, such as its screen and score; and
- it rewards the agent for its actions.

This section details the operation and main components of Gym Hero.

A. Gym Hero

Gym Hero was developed using Python and PyGame [24] graphics engine. It is divided into four parts: Notes and Buttons, Song, Score, and Game Loop.

1) Notes and Buttons: Gym Hero's screen (Fig. 2) has five colored buttons and a vertical lane where the notes slide from top to bottom until they reach the buttons or the end of the screen. The notes, which were created using PyGame Sprites, have the following attributes:

- Start – the note's initial tick, i.e., the moment it must be played;
- Type – "N" for a regular note, and "S" for a star note;
- Color – the note's color indicates the track to which the note belongs;
- Duration – the note's duration in ticks;
- Image – the note's 60 × 60-pixel representation rendered on the screen;
- Rect – storage of the note's current position on the screen (every Sprite in PyGame has a Rect attribute, which stores the position of the object on the screen).

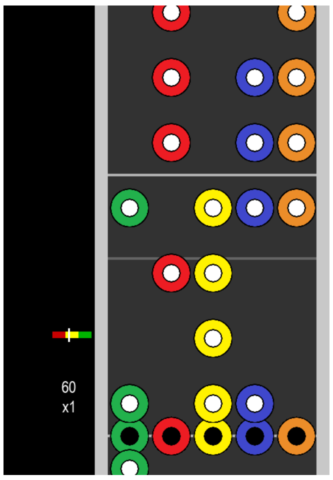


Fig. 2. Gym Hero's main screen.

Besides those attributes, a note also has an Update function, which updates its position on the screen by altering the value stored in Rect. Since the notes only move vertically, just the y coordinate is updated. The Update function also removes the note just after it is played or just after it reaches the end of the screen (when the player misses the note).

PyGame has a Sprite Group tool that allows calling the Draw and Update functions associated with each Sprite member of the Group. In Gym Hero, we use three instances of the Sprite Group tool: two for the notes and one for the buttons. To the first instance of Sprite Group, we add a new Sprite for each note that is created. Thus, all notes' positions are updated at every iteration. The second instance of Sprite Group is used to control which notes will be rendered. By limiting the number of Sprite members in this Sprite Group (we used 50 sprites), the rendering time of the game is reduced, because, even though all note positions are updated simultaneously, only the notes that are visible to the player will be rendered.

To distinguish the buttons from the notes visually, we painted their centers black (Fig. 3). For efficiency, the buttons are Sprite members of a separate Sprite Group (the third instance). However, only two attributes are present: Image and Rect. Since buttons neither move nor disappear, the Update function was not implemented for them.

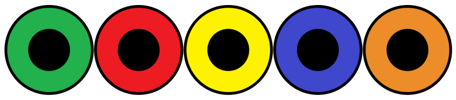


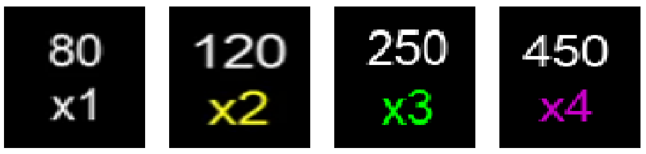
Fig. 3. Gym Hero's buttons.

2) Song: Gym Hero's song is stored in a text file, following the *chart* model presented in Section III. The audio is in *wav* format. However, in the experiments executed with autonomous agents, the audio was not used, since the agent uses only its vision to play the game. Initially, we extract the song metadata from the chart, and create a Song object that contains the following attributes:

- Offset – the number of ticks at the beginning of the song;
- Resolution – the number of ticks per beat;
- BPM – the number of beats per minute;
- Name – the name of the song;
- Guitar – the name of the audio file; and
- BPM Dict – the changes in BPM value.

Then, the notes are created according to the chosen difficulty level: Easy, Medium, Hard or Expert. The Easy level has only 3 buttons (Green, Red, and Yellow); the Medium level has 4 buttons (a Blue button is added); and the Hard and Expert levels have 5 buttons (an Orange button is included). The chart model has a section describing the notes for each difficulty level.

3) Score: In the Guitar Hero franchise, the player's goal is to get the highest score possible by the end of the song. The score is increased by 10 points for each note played correctly, however, this score is multiplied by a value that varies from 1 to 4, according to the number of consecutive notes played (Fig. 4). For every 10 consecutive notes played correctly, this number is increased by one. The player can also activate the special power, which doubles the multiplier value. In Gym Hero, we chose not to implement the special power, only the main score and multipliers were used, due to the increased complexity for the agent. We also added an option to decrease the score if the agent misses a note.



(a) Multiplier ×1. (b) Multiplier ×2. (c) Multiplier ×3. (d) Multiplier ×4.

Fig. 4. Score and Multiplier.

Alongside the Score and Multiplier, we also implemented the "Rock-Meter" (Fig. 5), which represents the crowd's satisfaction with the song's execution. It can be viewed as the life points of the player. Thus, when the player hits many notes, the Rock-Meter's score increases; on the other hand, when the player fails to hit the notes, the Rock-Meter's score decreases. If the score decreases to 0, the player is eliminated, and the song is ended earlier. We followed the original Rock-Meter design, a three-section bar divided into red, yellow and green, representing the increasing performance of the player.

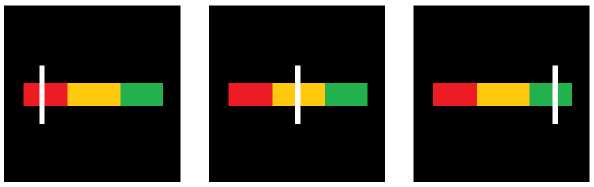

(a) Red RM. (b) Yellow RM. (c) Green RM.

Fig. 5. "Rock Meters" (RM).

4) Game Loop: The game loop is divided into two main phases:

- the update phase, in which all game values are updated (the score, the Rock-meter, the notes' positions, and the note's visibility; and
- the rendering phase, in which all visible objects are drawn on the screen.

During the update phase, the player's actions are registered. That is done in two different ways, depending on whether it is a human player or an agent player.

If the player is a human, the action is done using the keyboard and the keyboard events in PyGame. When an event happens, the game verifies weather the note intersects the central region of the button that corresponds to 60% of the button's area. The smaller that percentage is, the more accurate the player has to be. This is because the overlapping time of the note with that region becomes smaller when the central region of the button decreases. Thus, the challenge of the game increases. On the other hand, if the player is an agent, the action is registered through a five-element array representing the five buttons, from the green to the orange button. The array values are 1 or 0, indicating whether that button is pressed or not, respectively. The collision of the note with the button is considered true if the note intersects the central 60%-region of the button.

B. Gym's Environment

For Gym Hero to be capable of training Reinforcement Learning agents, we chose to implement a Gym environment because it is both easy to customize and popular, as presented in Section III. In this Section, we describe its main characteristics and their respective implementations. A sample code to run a random agent is shown in Fig. 6.

1) Action Space: this is the set of valid actions in the environment. In Gym Hero, the valid actions are the arrays described in Section IV-A4. Therefore, the Action Space was implemented as a MultiBinary object – an array of binary values available in the Gym Library. The size of the Action Space varies from 3 to 5, according to the chosen difficulty level. It is always the same as the number of buttons available in the game.

```
import gymhero_env

env = gymhero_env.GymHeroEnv()
obs = env.reset()
done = False
total_reward = 0.0

rewards = []
for i in range(1000):
    done = False
    while not done:
        action = env.actionspace.sample()
        obs, reward, done, = env.step(action)
        total_reward += reward

    rewards.append(total_reward)
    total_reward = 0.0
    obs = env.reset()
```

Fig. 6. Sample code to run a Random Agent.

2) Observation Space: this describes the structure of the valid observations about the environment. Gym Hero's environment was designed to use the game's screen as an agent's Observation Space. Therefore, the Observation Space was implemented as a Box object – a multidimensional matrix available in the Gym library. That matrix represents the colored game screen and its dimensions are the horizontal and vertical sizes of the screen, respectively, 500 and 720 pixels.

3) Step function: this function controls the game's dynamics. It advances a unit of time, which affects the time-dependent elements in the environment. This time advance is done through the Update function, described in Section IV-B. Its input parameter consists of an allowed action from the Action Space. The function returns a description of the results caused by the executed action in the environment, i.e., it returns the following updated values:

- Observation – an Observation Space object representing the updated state of the game;
- Reward – the reward received for executing the action;
- Done – a binary variable indicating whether or not the episode ended;
- Info – a dictionary containing the number of correctly played notes on an episode until the current step.

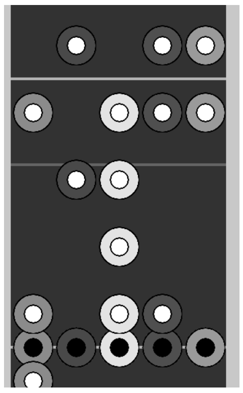
In Gym Hero, the rewards are: +10 points if the action led to a button hit; -10 points if the action led to a button miss; and 0 points if the action does not change the game state.

4) Reset function: this function starts a new episode, reverting the environment to its initial state, i.e., it starts the game loop presented in the Section IV-A4 and returns an initial observation, starting the cycle of agent-environment interaction.

C. The Agent

The main goal of the Reinforcement Learning agent developed in this paper is to learn how to play Gym Hero like human players. The only information available to the agent is the game's screen. Thus, the agent can be compared to human players more fairly, since it does not receive internal information of the environment.

Before being sent to the neural network, the input image passes through a preprocessing step that consists of: 1) transforming the colored image (Fig. 2) into a gray-scaled image, and cropping it (removing 150 pixels at the left and 150 pixel at the top) – Fig. 7a; and 2) resizing the cropped grayscaled image to a 54×48-pixel image – Fig. 7b.


(a) Cropped grayscaled image.

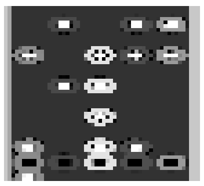

(b) Final image as seen by the agent.

Fig. 7. Preprocessing step.

D. *Neural Network's Architecture*

The neural network that controls the actions of the agent is divided into the following three main parts (Fig. 8 shows the summarized architecture):

1) Input: The input to the network is a grayscaled image with dimensions 54×48.

2) Feature Extraction: First, the image passes through two convolutional (conv) layers. The first conv layer has 32 feature maps of 27×24 pixels, obtained by filtering the input image with convolutional 4×4 kernels, and 2×2 strides. The second conv layer has 64 feature maps of 14×12, obtained by filtering the maps from the first layer also with convolutional 4×4 kernels, and 2×2 strides. Next, the 64 feature maps are flattened, i.e., they are arranged as a one-dimensional array with $64 \times 14 \times 12 = 10752$ elements; and each element is connected to 512 neurons of the next layer.

3) Output: The output layer contains one neuron for every possible action, and each neuron is fully connected with the 512 neurons of the previous layer.

E. *Hyperparameters*

We use the ReLU [25] activation function in both conv layers and the first fully connected layer. The hyperparameter values used were adapted from [23], due to the good results with a similar network architecture. We initialized the weights using Xavier's Initialization [26] and the model was trained using Adam's optimizer with a mini-batch of size 32.

To reduce the similarity between consecutive frames, we implemented an Experience Replay Memory [27]. In our implementation, we store the last thousand frames, and, at every iteration, the neural network receives a random sample from the memory.

At the training's outset, the agent explores the different actions for each state, since it does not have enough knowledge to decide what is the best action to execute. Therefore, we implemented an ϵ-decay policy, in which for each epoch the agent has a probability ϵ of executing a random action, instead of choosing the best one for that state. In our experiments, we followed the dynamics presented in [28], in which the ϵ value started at 1.0 for the first 2 epochs. Then it decayed linearly to 0.1 over the next 12 epochs, continuing at 0.1 until the end of the training.

V. Results and Discussion

In this section, we present the results obtained from the experiments using the Gym Hero's environment. The agents are analysed in accordance with the average reward, the average precision, and the iteration time. We also compare them with a random agent in each difficulty level, to serve as a baseline. Our goal is to verify if DRL agents can be effectively trained and evaluated using the environment proposed in this work.

A. *Training dynamics*

Each agent was trained for 20 epochs of 200 episodes using randomly generated songs. An episode ends either when the song finishes or when the agent fails. At the end of each epoch, the agent is tested for 100 episodes with randomly generated songs. At each new episode, a new chart was created by choosing random samples from the Action Space of the corresponding difficulty level every 48 ticks with a fixed size of 1920 ticks. This allows the agent to experience a greater number of different states while keeping a fair comparison between the agents. Three difficulty levels were used during the experiments: Easy, Medium, and Expert. Since the songs are randomly created according to the number of buttons, which is also the size of the Action Space, and both Hard and Expert have five buttons, the generated songs are equivalent. Therefore, we treat Hard and Expert as a single difficulty level.

The computer setting used in the experiments is shown in Table I.

TABLE I. Computer Hardware used in experiments.

Component	Name
CPU	Intel Core i7-10700 @ 2.9GHz
GPU	NVIDIA GeForce GTX 1070, 8GB VRAM
RAM	2 × HyperX Fury 8GB, 3000MHz

B. *Playing on Easy mode*

First, an agent was trained on the Easy difficulty level. At every epoch, we collected the minimum, maximum and average rewards. We also registered the average precision and the average duration of each episode. Here, precision is the number of note hits divided by the total number of notes in the song. The expected behavior of the agent is to wait for

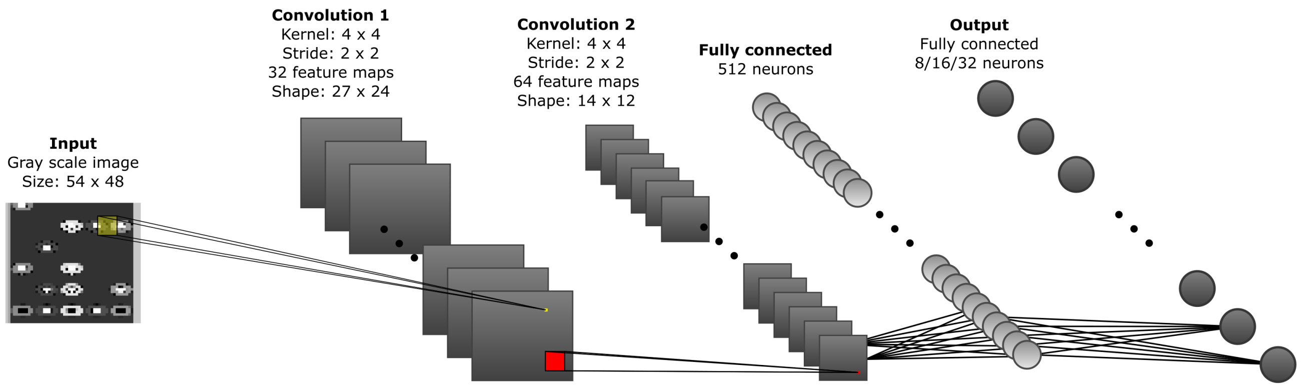
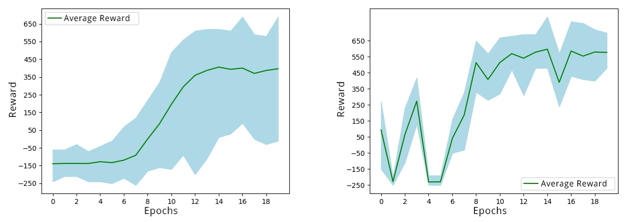


Fig. 8. Neural network's architecture.

a note to enter the collision region of the button, and then to send an action corresponding to the button needed to play that note. On average, charts used on the Easy level have 61 notes. Therefore, the mean reward value is around 610.

To achieve the highest score, the agent needs to remain alive until the end of the song. Therefore, it has to learn how to avoid pressing buttons when there are no notes. The agent learned to play on Easy mode, obtaining an average reward of 600 points (Fig. 9 and 10). The average reward on the last test run stands out, the agent obtained around 550 points, indicating that it learned not only when to press the buttons, but also when not to press them, since, if it had pressed, it would have obtained more negative penalties. It also obtained a high average precision score of 96.29% (Fig. 11), which means that the agent was able to play almost every note correctly.

One of the characteristics that indicates a good performance of the agent in Gym Hero is the iteration time, that is, how long each episode lasts. Due to Rock-Meter mechanics, if the agent misses too many notes, the song, and consequently the episode, may end prematurely. Therefore, the ascending curve in Fig. 12 points to a learning behavior. At the end of the tests, the mean iteration time while playing on Easy mode was 5 seconds, corresponding to the full length of the songs (Fig. 12).



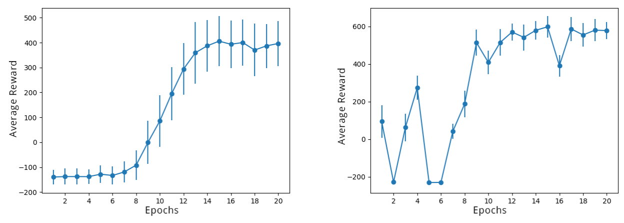
(a) Average training reward. (b) Average test reward.

Fig. 9. Average reward obtained by the agent trained on Easy mode.

C. Playing on Medium mode

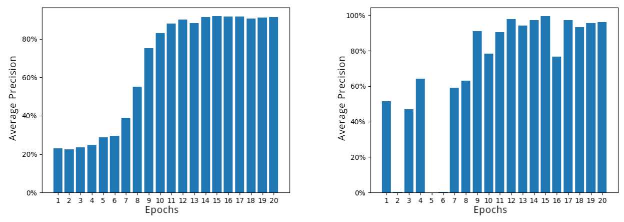
The second agent was trained on the Medium difficulty level according to the training dynamics described in Section V-A. In this difficulty level, the songs have approximately 82 notes. Therefore, an ideal agent would gather a mean reward value of 820 points.

The agent struggled more to learn how to play on Medium



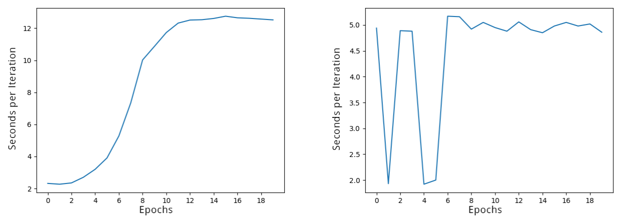
(a) Average and standard deviation training reward. (b) Average and standard deviation test reward.

Fig. 10. Average reward and standard deviation obtained by the agent trained on Easy mode.



(a) Average training precision. (b) Average test precision.

Fig. 11. Average precision obtained by the agent trained on Easy mode.



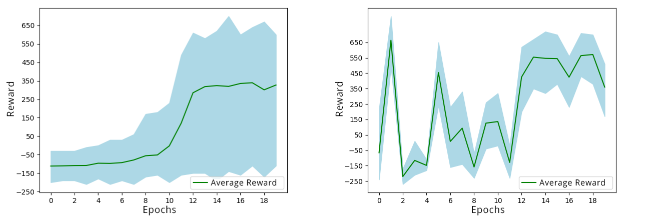
(a) Average training iteration time. (b) Average test iteration time.

Fig. 12. Average iteration time obtained by the agent trained on Easy mode.

mode. That happened because the Action Space's size doubles from Easy to Medium mode, going from 8 to 16 possible combinations. That decreases the probability of randomly hitting the right note from $\frac{1}{8}$ to $\frac{1}{16}$.

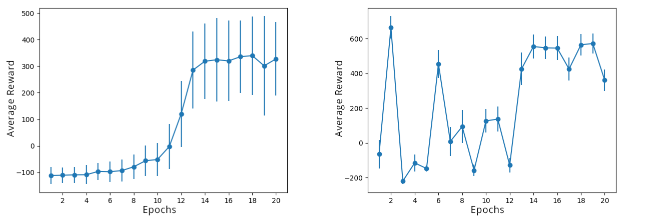
Despite that Action Space, the agent was able to learn how to play on Medium mode. Its average reward value was 360 points (Fig. 13), and Fig. 14 shows a greater variation through the much larger standard deviation bars in relation to the graph of the previous difficulty level (Fig. 10), confirming the struggle. However, notice a learning tendency on the the mean precision graphs (Fig. 15) due to the growth of the average test precision over the epochs, reaching 95.52% in the last two epochs.

The behavior of the agent trained on Easy difficulty level in relation to iteration time is repeated on Medium difficulty level. The generated songs for the Medium difficulty level have the same length of 5 seconds, and the agent mean iteration time was 4.5 seconds, indicating that it was able to finish most songs regularly (Fig. 16).



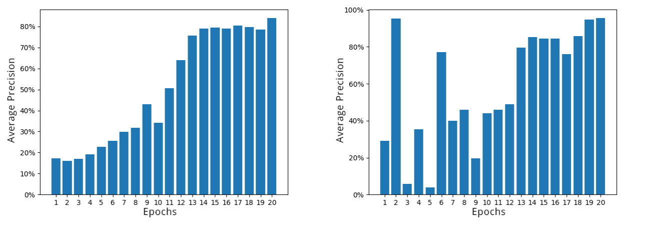
(a) Average training reward. (b) Average test reward.

Fig. 13. Average reward obtained by the agent trained on Medium mode.



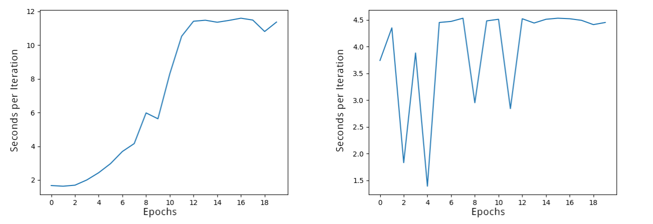
(a) Average and standard deviation training reward. (b) Average and standard deviation test reward.

Fig. 14. Average reward and standard deviation obtained by the agent trained on Medium mode.



(a) Average training precision. (b) Average test precision.

Fig. 15. Average precision obtained by the agent trained on Medium mode.

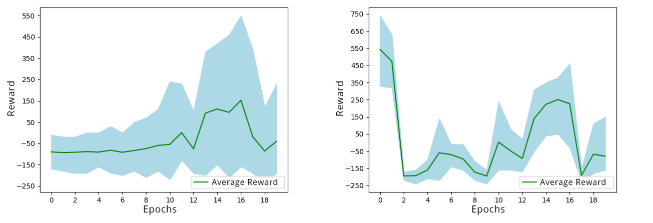


(a) Average training iteration time. (b) Average test iteration time.

Fig. 16. Average iteration time obtained by the agent trained on Medium mode.

standard deviation during training is larger in the second half of training, justifying the small upward tendency observed between epochs 10 and 18.

Fig. 19 corroborates the perception that the agent was unable to learn as its mean precision was just 7.22%, hardly exceeding 50% in the best episodes. The mean iteration time was 1.45 seconds (Fig. 20), implying that it was not able to reach the end of the episodes regularly, because the song would finish earlier.



(a) Average training reward. (b) Average test reward.

Fig. 17. Average reward obtained by the agent trained on Expert mode.

D. Playing on Expert mode

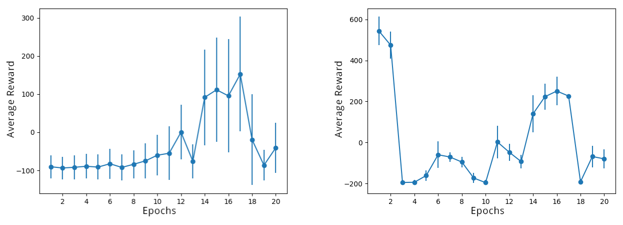
The agent trained on Expert mode did not learn how to play the game. In this difficulty, there are approximately 102 notes in the chart, so, an ideal agent should gather approximately 1020 reward points. Also, the number of possible actions doubles again, from 16 to 32, decreasing the probability of randomly hitting the right note to $\frac{1}{32}$.

The agent obtained an average reward value of -79.6, and the graphic in Fig. 17 does not show the ascending behavior that indicates learning. Also, Fig. 18 shows that the

E. Comparison with Random Agents

After we finished the evaluation of autonomous agents, three agents with random behavior were executed, one for each difficulty level. Those agents do not use a neural network for decision making, instead, they choose a random action within their Action Spaces. They played for 200 episodes, and we collected the average reward and the average precision for each difficulty level.

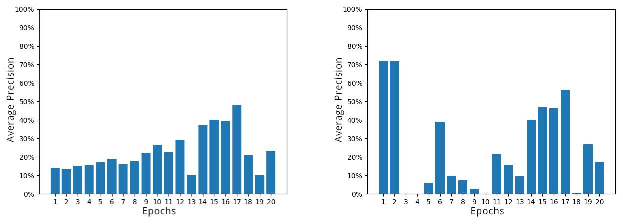
Table II presents a comparison between the average reward obtained by the autonomous and by the random agents. The



(a) Average and standard deviation training reward.

(b) Average and standard deviation test reward.

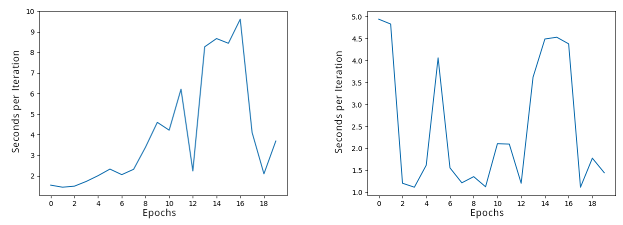
Fig. 18. Average reward and standard deviation obtained by the agent trained on Expert mode.



(a) Average training precision.

(b) Average test precision.

Fig. 19. Average precision obtained by the agent trained on Expert mode.



(a) Average training iteration time.

(b) Average test iteration time.

Fig. 20. Average iteration time obtained by the agent trained on Expert.

RL agent performed better in all three difficulty levels. That happened because of the low probability of the random agent executing the right action on a given state, since there is only one correct combination of buttons for each state and the number of possible actions grows exponentially at each new difficulty level.

Table III presents a comparison between the average precision of the autonomous and random agents. As seen, in Easy and Medium difficulty levels, the RL agent obtained over 95% precision, showing that it learned satisfactorily, but when tested on Expert mode, the average precision of the autonomous agent was lower than the random one. The final results are shown in Table IV.

TABLE II. Comparison between the average reward from the RL and Random agents.

Mode	RL Agent	Random Agent
Easy	$\mathbf{577,7 \pm 44,9}$	-133.8 ± 31.4
Medium	$\mathbf{360,1 \pm 60,8}$	-108.8 ± 30.0
Expert	$\mathbf{-79.6 \pm 46.1}$	-90.5 ± 32.4

TABLE III. Comparison between the average precision from the RL and Random agents.

Mode	RL Agent	Random Agent
Easy	96.29%	23.72%
Medium	95.52%	16.96%
Expert	7.22%	14.01%

TABLE IV. Result of 100 executions of each trained agent and the random agents.

Agent	Minimum Reward	Maximum Reward	Average Reward	Precision	Seconds / Iteraction
Easy	480	700	577.7 ± 44.7	96.29%	4.86
Medium	170	510	360.1 ± 60.8	95.52%	4.45
Expert	-160	150	-79.6 ± 46.1	17.30%	1.45
Random Easy	-270	-60	-133.8 ± 35.5	23.72%	1.56
Random Medium	-200	-20	-111.6 ± 31.4	16.96%	2.05
Random Expert	-180	10	-89.9 ± 32.4	14.01%	2.43

VI. Conclusion

In this work, we proposed a virtual learning environment for training and evaluation of Reinforcement Learning autonomous agents in rhythm games. We implemented a rhythm game similar to Guitar Hero with a Gym environment on top of it, allowing agents to interact with it. Furthermore, to validate the functioning of the environment within its proposed Reinforcement Learning task, an agent was developed using the Deep Reinforcement Learning technique called Deep Q-Networks.

As seen in Section V, agents trained on the Easy and Medium difficulty levels were able to learn how to play the game, obtaining results close to an ideal agent. However, the agent trained on the highest difficulty level was not able to win the game and obtained results comparable to a random agent. Thus, the developed environment was capable of evaluating autonomous Reinforcement Learning agents, which validates the goal of this work.

As future works, we intend to test different neural network architectures, since the NN highly influences the performance of the agent. We also intend to compare agents trained in Gym Hero competing against human players. In addition, as this is a Reinforcement Learning environment that contains multiple difficulty levels of the same game, agents that learn incrementally can be developed and tested using Curriculum Learning [29]. This method proposes to sequentially present more difficult and more complex tasks to the agents, which seems to be useful in Gym Hero.

References

[1] K. Müller, J. Schaeffer, and V. Kramnik, *Man Vs. Machine: Challenging Human Supremacy at Chess*. Russell Enterprises, Incorporated, 2018. [Online]. Available: https://books.google.com.br/books?id=k_w2uQEACAAJ

[2] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *ArXiv*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[9] T. Thorsen. (2010) Guitar hero tops $2 billion, activision blizzard earns $981 million in q1. [Online]. Available: https://www.gamespot.com/articles/guitar-hero-tops-2-billion-activision-blizzard-earns-981-million-in-q1/1100-6209327/

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI Gym,” *ArXiv*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[11] L. A. A. O. Vaz and M. Lamar, “Grybot: A didactic Guitar Hero robot player on FPGA,” *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pp. 118–127, 2015.

[12] P. Ganapati, “Guitar hero robot plays videogame with electronic precision,” Nov 2007. [Online]. Available: https://www.wired.com/2008/11/guitar-hero-rob/

[13] R. Mizrahi and T. Chalozin, “Guitar heronoid,” Mar 2007. [Online]. Available: http://guitarheronoid.blogspot.com/

[14] C. R. Breingan and P. Currier, “Development of a Rockband robot,” in *2012 Proceedings of IEEE Southeastcon*. IEEE, 2012, pp. 1–4.

[15] E. Adams, *Fundamentals of Game Design*, 2nd ed. USA: New Riders Publishing, 2009.

[16] Tarragon, “Guitar hero file formats,” Dec 2008. [Online]. Available: https://wiki.scorehero.com/GuitarHeroFileFormats

[17] M. Perry, “Guitar hero midi and vgs file details v1.3,” Oct 2008. [Online]. Available: https://wiki.scorehero.com/GHFileFormatDetails

[18] mikex5, “r/clonehero - a brief history of custom guitar hero chart formats and compatibility,” 2018. [Online]. Available: https://www.reddit.com/r/CloneHero/comments/8bkb0n/a_brief_history_of_custom_guitar_hero_chart/

[19] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[20] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[21] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “Vizdoom: A Doom-based AI research platform for visual reinforcement learning,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[22] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar *et al.*, “Unity: A general platform for intelligent agents,” *ArXiv*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[23] P. B. S. Serafim, Y. L. B. Nogueira, J. B. Cavalcante-Neto, and C. A. Vidal, “Deep reinforcement learning em ambientes virtuais,” in *Introdução a Realidade Virtual e Aumentada*, 3rd ed., R. Tori, M. S. Hounsell, C. G. Corrêa, and E. P. S. Nunes, Eds. Sociedade Brasileira de Computação - SBC, 2020, ch. 20, pp. 423–436. [Online]. Available: http://rvra.esemd.org/

[24] P. Shinners, “Pygame,” http://pygame.org/, 2000.

[25] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[26] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[27] L.-J. Lin, “Reinforcement learning for robots using neural networks,” Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 1993, uMI Order No. GAX93-22750.

[28] P. B. S. Serafim, Y. L. B. Nogueira, C. A. Vidal, J. B. Cavalcante-Neto, and R. F. Férrer Filho, “Investigating deep q-network agent sensibility to texture changes on FPS games,” in *Proceedings of the XIX Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2020, pp. 1–9.

[29] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.