# Recommender system for the billiard game

**Auteur :** El Mekki, Sélim
**Promoteur(s) :** Cornélusse, Bertrand
**Faculté :** Faculté des Sciences appliquées
**Diplôme :** Master en ingénieur civil électromécanicien, à finalité spécialisée en énergétique
**Année académique :** 2018-2019
**URI/URL :** http://hdl.handle.net/2268.2/6725

*University of Liège - Faculty of Applied Science*
*Academic year 2018-2019*

# Recommender system for the billiard game

*In fulfilment of the requirements for the Degree of Master in Electromechanical Engineering*

El Mekki Sélim

**Abstract**

This work studies how a recommender system for the billiard game can be treated as a reinforcement learning problem. The type of billiard game used is carom billiards. First, the geometry and physics of billiards are studied in order to make a simulator. Then, the simulator is designed following an event-based method simulation. This simulation method is the most suitable for this type of problem. The recommender system is then formalized as a reinforcement learning problem. It is then treated through different reinforcement learning algorithms such as Q-learning, Deep-Q-Learning and Deep Deterministic Policy Gradient. The results shows Deep deterministic policy gradient leads to the best agent behaviour compared to the other two techniques for the billiard game. Some tweaks are finally added to allow the recommender system to be able to find a solution regardless of the pool table state.

**Acknowledgements**

I would like to thank all the people who have contributed in some way to this work.

First of all, I would like to thank my academic promoters, Pr. Bertrand Cornélusse for his continuous follow-up and his suggestions in this work and Pr. Van Droogenbroeck for his advice on where and how to start this project. I would like to thank Mr. Quentin Gemine for accepting my internship at BLA and for giving me the opportunity to work on reinforcement learning based problems. Then, I thank my friend Selmane for giving me suggestions for this work. Finally, I would like to thank all my family for their support.

# Contents

# List of Figures

# 1 Introduction

The history of billiards goes back to the 14th century, where the first recognizable billiard was played outdoors. It was a game really similar to croquet. In the $15^{\text{th}}$ century, the first indoor pool table was owned by king Louis XI of France. Later, he refined and developed the rules, popularized the game, and it quickly spread among the French nobility. Nowadays, billiards sports are popular family of games of skill played around the world.

There exists several type of billiards games that can be separated into 3 categories:

- Pool: It is a category of billiards games played on a table with six pockets along the rails, in which the player has to push the balls. Here are some examples of Pool games: Eight-Pool, nine-ball, straight-pool.
- Games played on a snooker table which is billiards table with six pockets and dimensions that are different from Pool. This category of games include Snooker, English billiards and Russian pyramid.
- Carom billiards: this category of games is completely different from others since the table has not any pockets. The table also has different dimensions from others type of billiards games. This category includes: Carambole (also called Carom billiards or french Carom), cushion caroms, three-cushion billiards, artistic billiards and four-ball.

## 1.1 Carom Billiards

In this work, the type of billiards game studied is Carom billiards (french Carom). The game game is played with three balls and a billiards table without pockets.

A game of French billiards is played by two players, each of whom plays with his cue ball. The two cue balls are distinguished by their color: one is white and the other is yellow. With their ball, the players must simultaneously touch the other two balls. The main goal of the game is to make the longest possible series of points. As long as the player is successful in the moves, he continues. When he misses the point, it is his opponent who plays again. The first of the players to score 300 points wins the game but, in general, this number of goal points can be adjusted to a lower value according to the level of the players and they play a limited number of turns.

In order to decide which player will start the game, the two opponents simultaneously play their ball, placed at the height of the starting marked positions and must hit the opposite small rail and get as close as possible to the rail behind the starting position. The player closest to the rail chooses: either he starts or he decides to let his opponent play the first move. The complete and detailed rules can be found in [1].

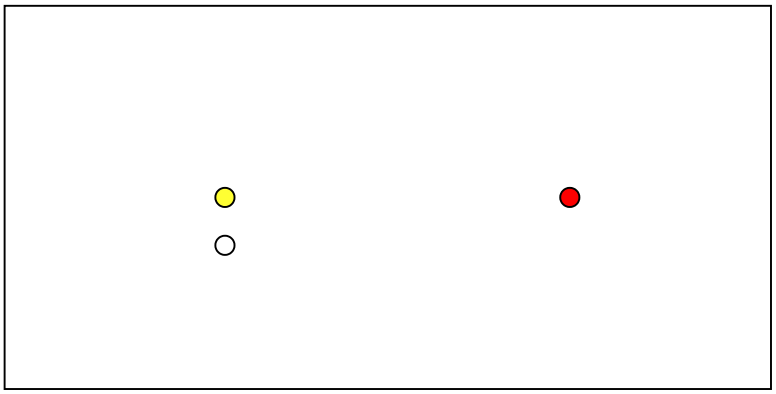


Figure 1: Carom table before the first strike.

## 1.2 State of the art

First, the physics involved in billiards was analyzed by Coriolis [2]. The analysis made is very advanced and often serves as a reference for the current studies on billiards. Later, as knowledge of physics had improved considerably, Coriolis' work was updated in some books. Examples include Petit [3] or Marlow [4], books that focus on the physical laws involved in billiards.

On the simulator side, there are a few. We can mention for example, FooBillard, Coriolis3D or Fastfiz. The reasons to create a simulator in this work are: first, the fact that most of the known simulators are not open source and second, the fact of doing it in python, the language in which most libraries related to machine learning and reinforcement learning are located.

As for billiard agents, there are already some, such as PickPocket [5], PoolMaster [6], DeepGreen [7], RoboShark [8].

### 1.2.1 Motivation

The creation of agents to play games is one of the most important areas of research in reinforcement learning. Games are suitable devices for reinforcement learning because they provide simple environment in which agents can quickly interact with and train on and in several cases, they can be easily simulated. In the case of billiards, a lot of work has been done in the application of AI techniques to this game.

Traditionally, AI techniques used for this game include research and heuristic methods. More recent works also used image data of the game and supervised learning. While both types of techniques have worked well in their respective settings, they both have their issues. Heuristics and search require that the agent has a full understanding and information of its environment and is hard to be generalized to unknown situations. Image based methods can be generalized but they require very large amounts of data and computation to train.

Reinforcement learning avoid these issues. It has lower data and computational needs than supervised learning (deep learning in particular). In addition to this, RL could still be generalized to unseen states of environments contrary to heuristics which have to be tuned. Moreover, little work is done for French billiards. Most of the agents were made for 8-pool.

# 2 The Physics Of Billiards

## 2.1 Ball Motion

Contrary to what one might think a ball trajectory is not always a straight motion. A ball can have different types of motion on the pool table: a sliding motion or a rolling motion. This is due to the friction force with the table which is the only unbalanced force. The motion of the cue ball when it is struck begins by sliding. After a while, the ball change its motion to rolling.

Considering a ball with a radius $R$, these motions are classified using the ball contact point relative velocity $\mathbf{u}$ with the table [1]:

$$\mathbf{u} = \mathbf{v} + R\hat{\mathbf{e}}_z \wedge \boldsymbol{\omega} \tag{1}$$

where $\mathbf{v}$ and $\boldsymbol{\omega}$ are respectively the linear speed and the angular velocity of the ball and $\hat{\mathbf{e}}_z$ is the unit normal vector of the table.

The ball is in a sliding state when $\mathbf{u} \neq 0$. When the ball is in rolling state, the ball travels completely its perimeter per revolution, this happens when $\mathbf{u} = 0$.

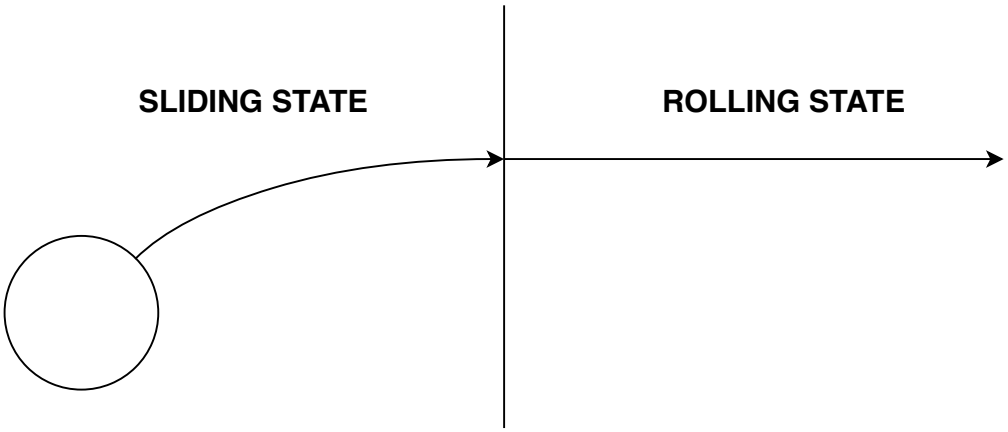



Figure 2: Ball Motion.

### 2.1.1 The sliding state

Sliding starts when $\mathbf{u} \neq 0$. In this state the velocities and the displacement equations are given by [9] :

$$\mathbf{v} = \mathbf{v_0} - \mu_s g t \hat{\mathbf{u}} \tag{2}$$

$$\boldsymbol{\omega} = \boldsymbol{\omega_0} + \frac{5}{2R}\mu_s g t \hat{\mathbf{e}}_z \wedge \hat{\mathbf{u}} \tag{3}$$

$$\mathbf{P} = \mathbf{P_0} + \mathbf{v_0} t - \frac{1}{2}\mu_s g t^2 \hat{\mathbf{u}} \tag{4}$$

Where $\mathbf{v_0}$ is the initial speed, $\boldsymbol{\omega_0}$ is the initial rotational speed, $\mathbf{P}$ is the ball position and $\mu_s$ is the sliding friction coefficient.

When sliding, the friction coefficient is higher and the trajectory may become curvilinear. This is the case for example when the ball is struck with an elevated cue.

### 2.1.2 The rolling state

The ball begins its rolling motion a time $\tau_{slide}$ after being struck (when $\mathbf{u} = 0$), the velocities and displacement equations are given through [9]:

$$\mathbf{v} = \mathbf{v_0} - \frac{5}{7}\mu_r g t \hat{\mathbf{v}} \tag{5}$$

$$\boldsymbol{\omega} = \frac{\hat{\mathbf{e}}_z \wedge \mathbf{v}}{R} \tag{6}$$

$$\mathbf{P} = \mathbf{P_0} + \mathbf{v_0} t - \frac{5}{14}\mu_r g t^2 \hat{\mathbf{v}} \tag{7}$$

Where $\mu_r$ is the rolling coefficient. In this case, the time $t$ begins when the sliding ends and the rolling begins. The ball becomes stationnary when the rolling state ends. This will occur after a time $\tau_{rolling}$. The rolling trajectory is always a straight line and the friction coefficient is smaller.

### 2.1.3 The spinning state

As seen above, the rotational speed vector is always perpendicular to the z axis and therefore has no component along this axis. However, in reality this is not the case, when one pulls on the side of a ball, it will rotate at a speed whose component according to z is not zero. This type of spin movement is called spin along the z axis, it appears when you play a "left english" or "right english" move.

Therefore, this type of movement must be added to the previous movements as a combination. This means that the previously described states (sliding and rolling)

can be combined with spinning or not depending on the value of the component along the z axis of the rotation speed.

The component along the z-axis of the rotational speed is described by [10]:

$$\omega_z(t) = \omega_z(0) - \frac{5g\mu_{sp}}{2R}t \tag{8}$$

Where $\mu_{sp}$ is the spinning friction coefficient.

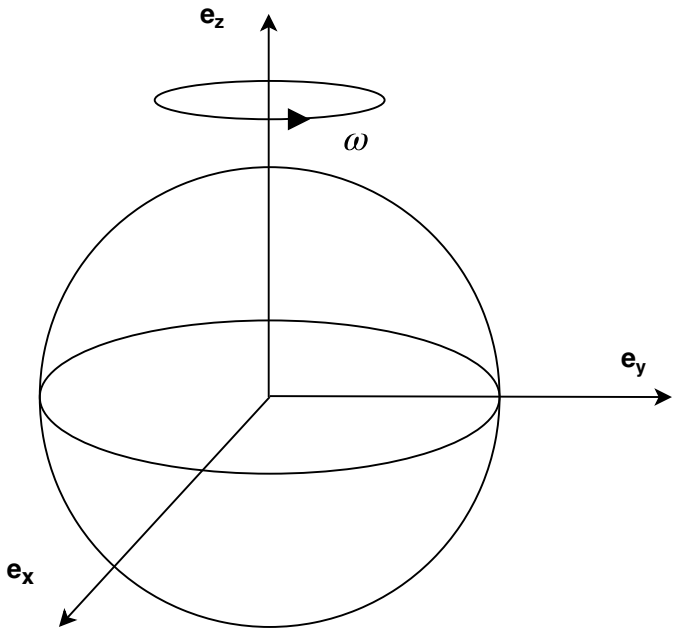



Figure 3: Ball in spinning state.

## 2.2 Collisions

Collisions are an important part of the billiard game. It is therefore essential to describe these collisions precisely so that the simulator can produce results that are close to reality.

There are three types of collision for carom billiards: the collision between a ball and the cue, the collision between two balls and the collision between a ball and a side rail. These are described below.

### 2.2.1 Cue-Ball Collision

The goal here is to be able to transform the input variables applied to the cue into initial ball variables (initial linear velocity and initial rotational velocity).

Assuming a relatively short collision time[1] and assuming that the collision occurs at a single point, it is possible to model the cue inputs into 5 distinct variables:

- $a$ and $b$, respectively, the horizontal and the vertical position of the impact point. Theses inputs are the sources of side, back and top spins.
- $\theta$ the elevation angle of the cue with the horizontal plane
- $\phi$ the angle the cue make with the vertical plane as can be seen in figure 6.
- $V$ the cue speed just before the stroke.

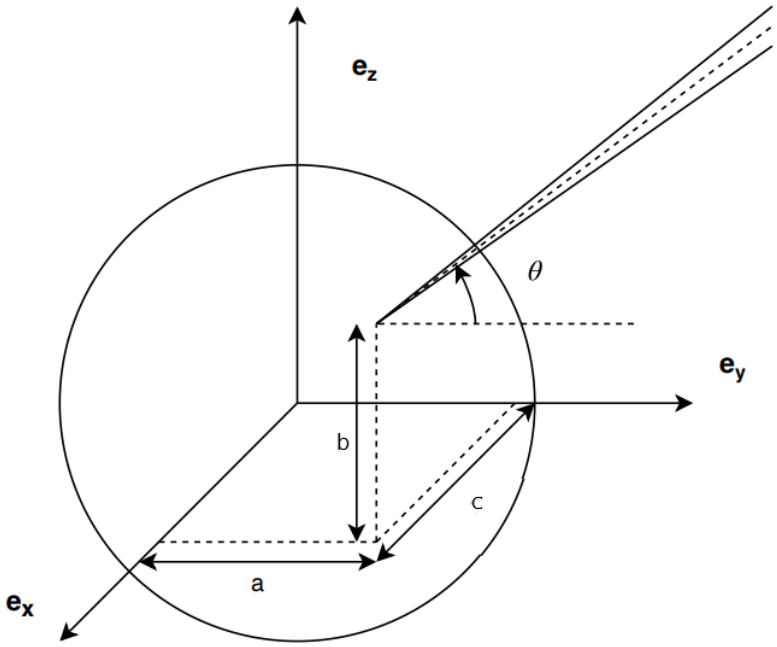



Figure 4: Cue-Ball collision.

First, the value of the force applied by the cue to the ball can be calculated by using the conservation of linear momentum and the conservation of energy [11]:

$$F = \frac{2mV}{1 + \frac{m}{M} + \frac{5}{2R^2}(a^2 + b^2\cos^2\theta + c^2\sin^2\theta - 2bc\cos\theta\sin\theta} \tag{9}$$

[1]empirically determined, the collision duration is approximately 200 $\mu$sec at a speed of 1 m/s according to [4].

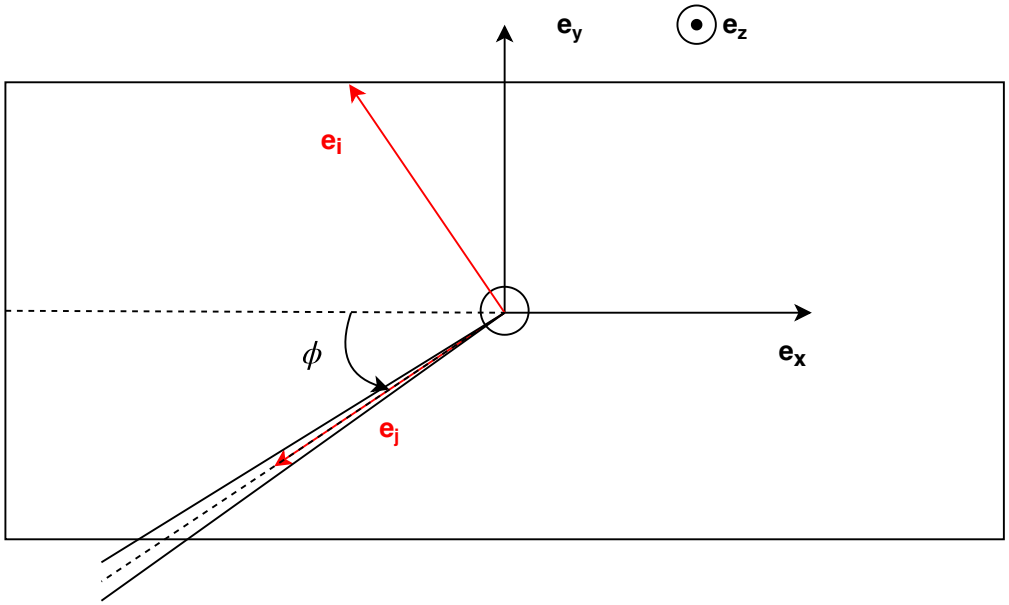



Figure 5: $\phi$ input parameter and reference frames.

Where $m$, $M$ and $c$ are respectively the ball mass, the cue mass and the x coordinate of the impact point. $c$ is determined by using twice the Pythagorean theorem.

$$c = \sqrt{R^2 - a^2 - b^2} \tag{10}$$

by considering a reference frame $(\hat{\mathbf{e}}_i, \hat{\mathbf{e}}_j, \hat{\mathbf{e}}_z)$ such that the cue direction is parallel to $\hat{\mathbf{e}}_j$.

Using the assumption of a very short duration collision, the force exerted by the cue on the ball can be considered as a perfectly elastic impulse. Then, by integrating Newton's second law, it is possible to express the initial linear velocity of the ball in this reference frame [11][12].

$$\mathbf{v} = \begin{pmatrix} 0 \\ \dfrac{-F}{m}\cos\theta \\ \dfrac{-F}{m}\sin\theta \end{pmatrix} \tag{11}$$

When the ball is struck by the cue, it also receives an initial angular velocity. The values of $a$ and $b$ strongly impact this angular velocity. Indeed, as explained above, when hitting on the right at the center of the ball ($a > 0$) or on the left ($a < 0$) it creates a side spin that influences the component along the z axis of the angular

velocity. The same is true for $b$, when you hit above the center of the ball ($b > 0$) or below ($b < 0$), it creates a top spin or back spin that impacts the component along the x axis of the angular velocity. These moves have particular names in the game of billiards: the first two correspond to the "English", the third corresponds to the "Follow" and the last is the "Draw".

By using a method similar to that used to express velocity as a function of force, i.e. by integrating Newton's second law in rotation, it is possible to find the rotational speed in the reference frame $(\hat{\mathbf{e}}_i, \hat{\mathbf{e}}_j, \hat{\mathbf{e}}_z)$.

$$\boldsymbol{\omega} = \frac{1}{I} \begin{pmatrix} -cF\sin\theta + bF\cos\theta \\ aF\sin\theta \\ -aF\cos\theta \end{pmatrix} \tag{12}$$

Where $I$ is the moment of inertia of the ball. The moment of inertia of a sphere is equal to $\frac{2mR^2}{5}$. However, for the creation of the simulator, it is preferable to work with a fixed reference frame $(\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z)$. The center of this one is the center of the table, at the height of a radius above the pool table. The x-axis is directed towards the right rail and perpendicular to the right rail while the y-axis is directed towards the top rail and perpendicular to it. The reference frame can be seen in the figure 6.

To do this, the previous reference frame must be rotated by $\phi + \frac{\pi}{2}$. The 3-dimensional rotation matrix $R_z(\alpha)$ around the z-axis is used.

$$\mathbf{R_z}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{13}$$

Replacing $\alpha$ by $\phi + \frac{\pi}{2}$, the matrix becomes:

$$\mathbf{R_z} = \begin{pmatrix} -\sin\phi & -\cos\phi & 0 \\ \cos\phi & -\sin\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{14}$$

The linear speed and the rotational speed are now calculated in the fixed reference frame $(\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z)$.

$$\mathbf{v}' = \mathbf{R_z}\mathbf{v} = \begin{pmatrix} -\sin\phi & -\cos\phi & 0 \\ \cos\phi & -\sin\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{-F}{m}\cos\theta \\ \frac{-F}{m}\sin\theta \end{pmatrix} = \begin{pmatrix} \frac{F}{m}\cos\theta\cos\phi \\ \frac{F}{m}\cos\theta\sin\phi \\ \frac{-F}{m}\sin\theta \end{pmatrix} \tag{15}$$

$$\boldsymbol{\omega}' = \mathbf{R}_z\boldsymbol{\omega} = \frac{1}{I}\begin{pmatrix} -\sin\phi & -\cos\phi & 0 \\ \cos\phi & -\sin\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} -cF\sin\theta + bF\cos\theta \\ aF\sin\theta \\ -aF\cos\theta \end{pmatrix}$$
$$= \frac{1}{I}\begin{pmatrix} cF\sin\theta\sin\phi - bF\cos\theta\sin\phi - aF\sin\theta\cos\phi \\ -cF\sin\theta\cos\phi + bF\cos\theta\cos\phi - aF\sin\theta\sin\phi \\ -aF\cos\theta \end{pmatrix} \quad (16)$$

### 2.2.2 Ball-Ball Collision

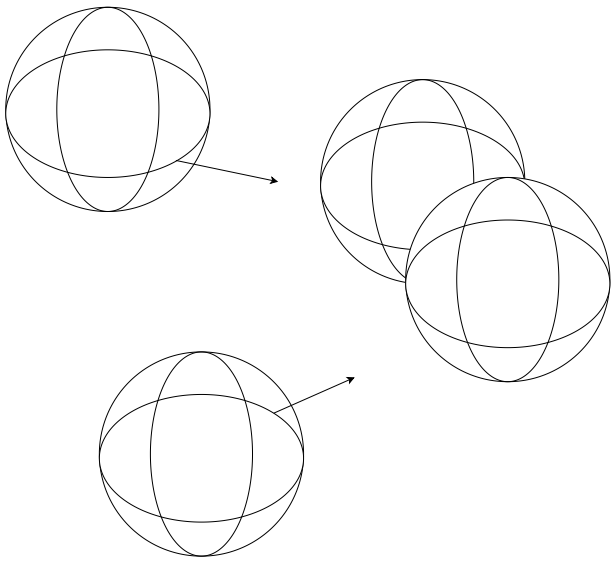


Figure 6: General Ball-Ball Collision.

In this section, the objective is to determine the ball variables (speed and rotation speed) immediately after the collision, given these same variables just before the impact. General collision between two rigid bodies have been analyzed and put into equations [3]. Then, considering two spheres of same radius and mass

instead of general rigid bodies, these equations can be simplified to

$$\begin{cases} \mathbf{v_1'} = \mathbf{v_1} + \dfrac{P}{m}\hat{\mathbf{n}} - \dfrac{1}{7}\mathbf{V_{c,i}} \\ \mathbf{v_2'} = \mathbf{v_2} + \dfrac{P}{m}\hat{\mathbf{n}} - \dfrac{1}{7}\mathbf{V_{c,i}} \\ \boldsymbol{\omega_1'} = \boldsymbol{\omega_1} + \dfrac{5}{7R}\left(\hat{\mathbf{n}} \wedge \dfrac{\mathbf{V_{c,i}}}{2}\right) \\ \boldsymbol{\omega_2'} = \boldsymbol{\omega_2} + \dfrac{5}{7R}\left(\hat{\mathbf{n}} \wedge \dfrac{\mathbf{V_{c,i}}}{2}\right) \end{cases} \tag{17}$$

where $\hat{\mathbf{n}}$ is the unit vector going from the collision contact point to the center of ball 1, $\mathbf{P}$ is the percussion vector coming from Coulomb Law for solid friction and $\mathbf{V_{c,i}}$ is the sliding speed vector of the contact point.

$$\mathbf{P} = P\left(\hat{\mathbf{n}} - \mu\frac{\mathbf{V_{c,i}}}{\|\mathbf{V_{c,i}}\|}\right) \tag{18}$$

$$P = km(\mathbf{v_2} - \mathbf{v_1}) \cdot \hat{\mathbf{n}} \tag{19}$$

where $\mu$ is the friction coefficient between the two balls and $k$ is constant related to the ball coefficient of restitution $e$ by:

$$k = \frac{1+e}{2} \tag{20}$$

The sliding speed vector of the contact point is determined, according to [13], by :

$$\mathbf{V_{c,i}} = (\mathbf{v_1} - \mathbf{v_2}) - [(\mathbf{v_1} - \mathbf{v_2}) \cdot \hat{\mathbf{n}}]\,\hat{\mathbf{n}} + R\hat{\mathbf{n}} \wedge (\boldsymbol{\omega_1} + \boldsymbol{\omega_2}) \tag{21}$$

A simple example where the second ball is at rest is provided in figure 7.

### 2.2.3 Rail-Ball Collision

In the case of a rail-ball collision, it is assumed that the mass of the rail is infinitely greater than that of the ball. In addition, the speeds for the rail are considered to be zero. As a result, the collision equations of two rigid bodies [3] are simplified in [13]:

$$\begin{cases} \mathbf{v_1'} = \mathbf{v_1} + \dfrac{P}{m}\left(\hat{\mathbf{n}} - \mu\dfrac{\mathbf{V_{c,i}}}{\|\mathbf{V_{c,i}}\|}\right) \\ \boldsymbol{\omega_1'} = \boldsymbol{\omega_1} + \dfrac{5P\mu}{2Rm}\left(\hat{\mathbf{n}} \wedge \dfrac{\mathbf{V_{c,i}}}{\|\mathbf{V_{c,i}}\|}\right) \end{cases} \tag{22}$$

Where, in the case of a ball-rail collision, $\hat{\mathbf{n}}$ is the unit vector perpendicular to the rail and going from it to the ball. $P$ can be retrieved from (19). However the constant $k$ in (19) is calculated through (23) for a rail-ball collision.

$$k = 1 + e \tag{23}$$

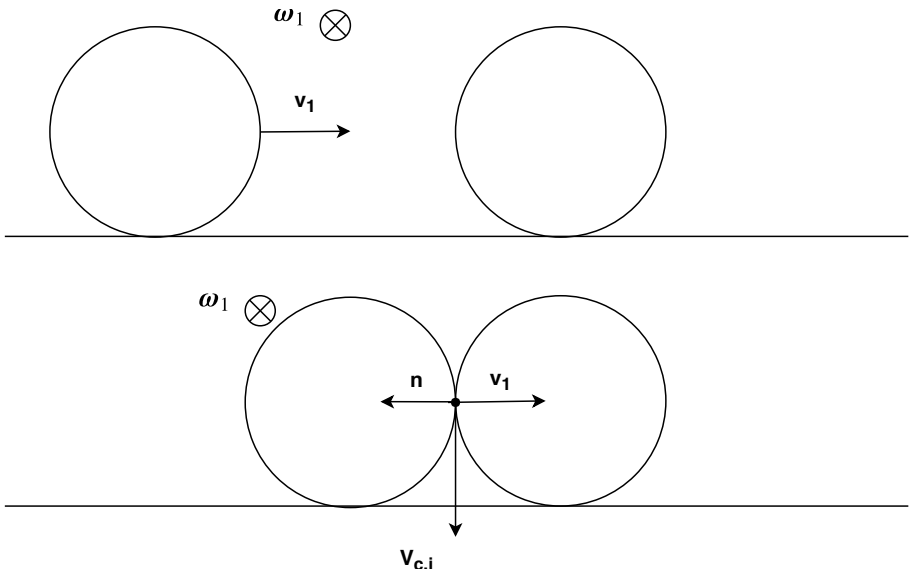



Figure 7: Sliding speed vector example with one ball at rest.

## 2.3 Billiard Geometry

The geometry of the table and the initial state is given in figure 8. The introduction of constant variables like $l$ and $L$ are important for the collision detection with rails which are introduced in the simulation modeling section.

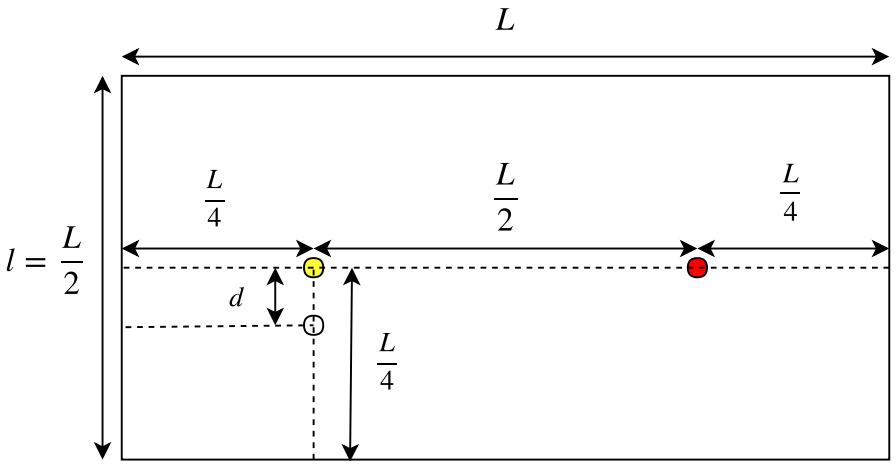



Figure 8: Billiard table geometry and initial state geometry.

| Simulator parameters | Values |
|---|---|
| $L$ [m] | 2.54 |
| $l$ [m] | 1.27 |
| $d$ [m] | 0.163 |
| $g$ [m/s$^2$] | 9.81 |
| $\mu_s$ | 0.2 |
| $\mu_r$ | 0.016 |
| $\mu_{sp}$ | 0.044 |
| $m$ [kg] | 0.21 |
| $M$ [kg] | 0.54 |
| $R$ [m] | 0.0305 |
| $e$ | 0.3 |

Table 1: Geometric and physical parameters used for the simulator.

# 3 Simulation Modeling

There are several ways to simulate a pool game. The first method that comes intuitively is simply to simulate the game with a constant time step $\Delta t$. At Each time step, objects move using the previous displacement equations and their current velocity. At the end of the time step, the simulator detects if some event such as a collision happened during this time step.

Another way is to use a simulation by event prediction [11][12]. This type of simulation does not use a time step. It directly predicts the next event that will happen using the analytical and deterministic equations of physics.

The different events that can happen and disturb the physics are the following:

- The starting cue stroke
- a ball passing from a sliding state to a rolling state
- a ball passing from a rolling state to a stationary state
- a ball passing from a spinning state to a non-spinning state
- a collision between two balls
- a collision between a ball and a rail

**Simplifying Assumption**

The main simplifying assumption made for the simulation is the freezing of the $z$ component of the balls position in order to avoid some flying states since the goal is to make a recommender system for simple shots and not advanced shots. A flying state, would greatly complicate the problem.

## 3.1 Intuitive Method

Considering the states of the balls (sliding, rolling or stationary), the displacement equations are provided in section 2.1. Then, they are used by discretizing the time with a constant time step. For each time step simulated, the simulator checks if an event will occur at the end of this timestep. An example, where the ball moves using its equations until a collision is detected, is shown in figure 9.

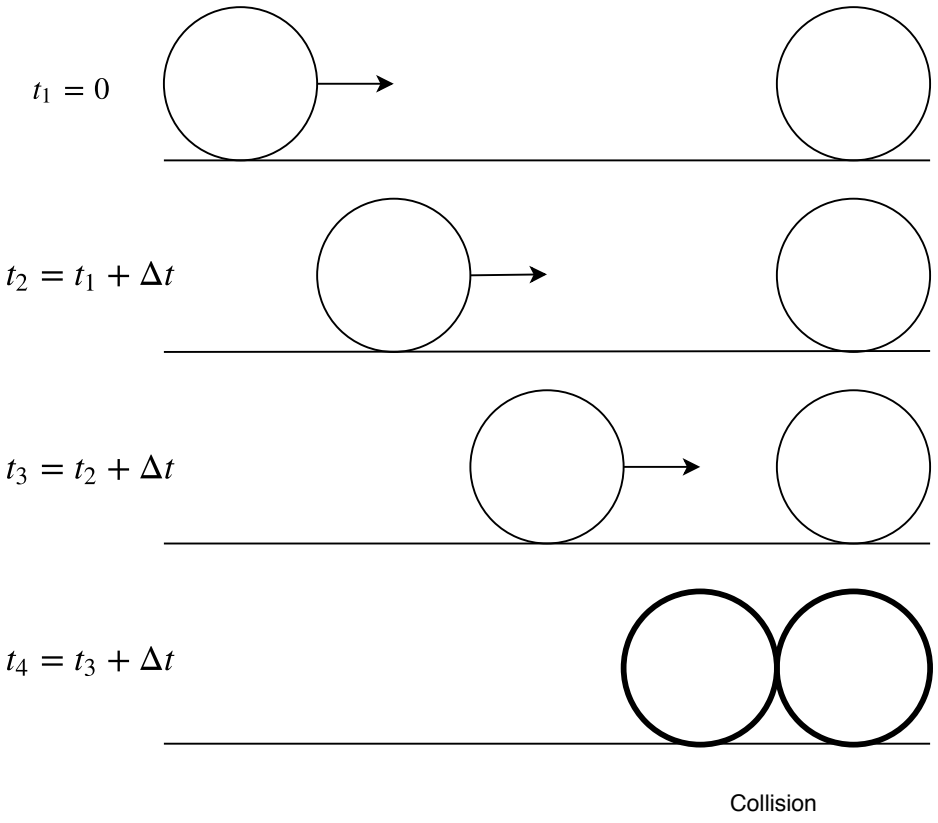



Figure 9: Simulation with an intuitive method.

Due to its logic and simplicity of implementation, this method is used in most simulations of physical phenomena. However, in the case of billiards, this type of simulation should be avoided for various reasons [13]. When discretizing, the speeds and accelerations are considered constant during a particular timestep. This leads to small numerical errors at each time step. But the main approximation issue with this simulation method is that choosing a constant time step leads to large errors when an event is detected. Indeed, an event will never happen exactly at a moment $t_x + \Delta t$. The figure 9 is not totally correct at the last timestep and should be more like the figure 10.


Figure 10: Intuitive method drawback.

Another approximation issue, is when two events happen in a same time step. For example a ball collides with another one and a rail at the same time as can be seen in figure 11.

Figure 11: A ball collides with both another one and a rail during a same timestep.

There are solutions to these approximation problems. However, to apply reinforcement learning to billiards, the simulator must be able to play an extremely large number of games. Learning by simulating at "real" speed can be very long. The number of time steps should therefore be minimized in order to simulate more quickly. However, increasing the time step leads to much higher approximation errors.

For these various reasons, this type of simulation is avoided for billiards. It is therefore the event-based simulation that will be applied to our model. This is described in the following section.

## 3.2 Event-based Method

This method described by [11][12], is a different way to simulate the game. In this case, there is not any time step. The events are predicted in advance and the game is simulated through events. For example, here is a simple game described through events: Cue strike, Sliding to rolling, Rolling to stationary. Between these events, the ball motion equation does not change: after the cue strike, the ball is in sliding state until the next event which is "Sliding to rolling". After this event, the ball starts to roll using the equations relative to this motion until the stationary state. Using this simulation method allows to avoid simulation at each time step which really speeds up the simulation since its simulated only through events.

Using this method, the simulation can only be done if you know about a given event, the next event that must happen. That is, find the time when the next event will take place. Thus it is possible to apply the equations of motion given by the first event for a period of time calculated in advance. And it is after this calculated time that the next event can take place. Calculating the time at which the next event will occur gives the simulator an exact solution contrary to an intuitive method which gives an approximate solution (in which a ball never collides at $t_x + \Delta T$ for example). This type of simulation can be done only under 3 assumptions [13]:

1. All types of event can be predicted.
2. Two or more events cannot be simultaneous.
3. All events are instantaneous.

All assumptions are met, but for some particular moves the simulation fails. For example, at the beginning of a game. If one strikes the ball with an angle $\phi = \frac{3\pi}{4}$ the ball will collide with the left rail and the down rail at exactly the same time since the initial distance of the ball to the left and down rail is exactly the same as can be seen in figure 12. But this happens only for particular moves and never happens in reality. For example, setting $\phi = 134.99°$ instead of $\phi = 135°$ solves the problem.
As said previously, to implement this simulator, it is mandatory to find a way to calculate the instant at which each type of event can occur. This is done by solving analytic equations of motion in the next section.

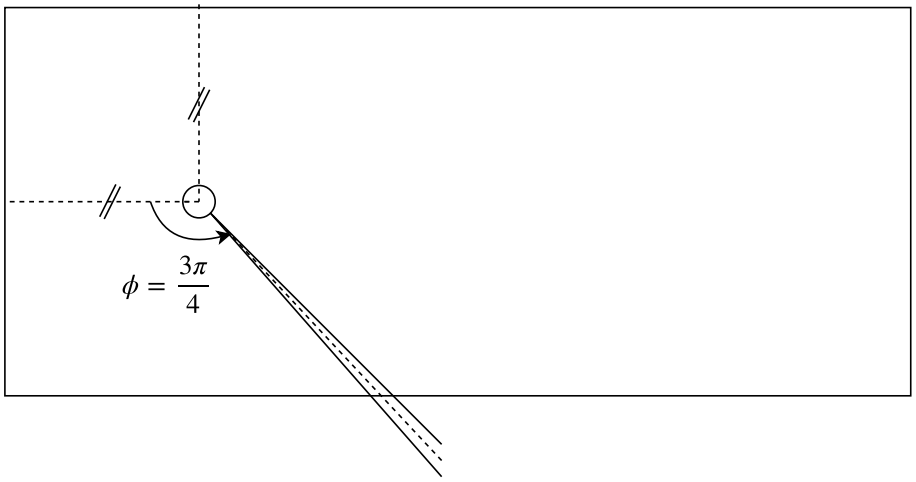



Figure 12: Simultaneous events move.

## 3.3 Event prediction solver

### 3.3.1 Cue strike event

The cue strike event is trivial and always occurs at time $t = 0$ for any game. Directly after this event, the ball is in a sliding state until the next event.

### 3.3.2 Sliding to rolling event

The sliding state ends when $\mathbf{u} = 0$. Using equations (2) and (3) in equation (1), the time associated to $\mathbf{u} = 0$ can be found.

$$\begin{aligned}
\mathbf{u} &= \mathbf{v} + R\hat{\mathbf{e}}_z \wedge \boldsymbol{\omega} \\
&= \mathbf{v_0} - \mu_s g t \hat{\mathbf{u}} + R\hat{\mathbf{e}}_z \wedge \left( \boldsymbol{\omega_0} + \frac{5}{2R}\mu_s g t \hat{\mathbf{e}}_z \wedge \hat{\mathbf{u}} \right) \\
&= \underbrace{\mathbf{v_0} + R\hat{\mathbf{e}}_z \wedge \boldsymbol{\omega_0}}_{\mathbf{u_0}} - \mu_s g t \hat{\mathbf{u}} + R\hat{\mathbf{e}}_z \wedge \left( \frac{5}{2R}\mu_s g t \hat{\mathbf{e}}_z \wedge \hat{\mathbf{u}} \right) \\
&= \mathbf{u_0} - \mu_s g t \hat{\mathbf{u}} + \frac{5}{2}\mu_s g t \left[ \hat{\mathbf{e}}_z \wedge (\hat{\mathbf{e}}_z \wedge \hat{\mathbf{u}}) \right] \\
&= \mathbf{u_0} - \mu_s g t \hat{\mathbf{u}} + \frac{5}{2}\mu_s g t \left[ \underbrace{(\hat{\mathbf{e}}_z \cdot \hat{\mathbf{u}})}_{=0} \hat{\mathbf{e}}_z - \underbrace{(\hat{\mathbf{e}}_z \cdot \hat{\mathbf{e}}_z)}_{=1} \hat{\mathbf{u}} \right] \\
&= \mathbf{u_0} - \frac{7}{2}\mu_s g t \hat{\mathbf{u}}
\end{aligned} \tag{24}$$

The sliding ends at a time $\tau_{slide}$ that can be found by solving (24) for $\mathbf{u} = \mathbf{0}$.

$$\tau_{slide} = \frac{2\,\|\mathbf{u_0}\|}{7\mu_s g} \tag{25}$$

### 3.3.3 Rolling to stationary event

The ball becomes stationary when $\mathbf{v} = 0$. Considering $t = 0$ is the time when the rolling begins, the time $\tau_{roll}$ can be found by solving equation (6) for $\mathbf{v} = \mathbf{0}$.

$$\tau_{roll} = \frac{7\,\|\mathbf{v_0}\|}{5\mu_r g} \tag{26}$$

### 3.3.4 Spinning to non spinning event

The spinning state ends when $\omega_z = 0$. Considering $t = 0$ is the time when the spinning begins, the time $\tau_{spin}$ can be found by solving equation (8) for $\omega_z = 0$.

$$\tau_{spin} = \frac{2R\omega_z(0)}{5\mu_{sp} g} \tag{27}$$

### 3.3.5 Ball-rail collision event

The goal here is to find the time the ball will collide with a rail. There exist two cases, the ball can either be in sliding state or in a rolling state.

- **The ball is in sliding state**

  Using the displacement equation for a ball in sliding state (4), the time $\tau_{rb,coll}$ can be found. To detect the collision with the right or the left rail, the $x$ component of equation (4) is used.

  $$P_x = P_{0,x} + v_{0,x}t - \frac{1}{2}\mu_s g t^2 \widehat{u}_x \tag{28}$$

  The ball collides with the right rail if $P_x = \frac{L}{2} - R$ since $P_x$ is the position of the ball center. Hence, the time it collides with the right rail $\tau_{rb,coll}$ can be found by solving the quadratic equation (33) and taking the smallest positive solution:

  $$-\frac{1}{2}\mu_s g \tau_{rb,coll}^2 \widehat{u}_x + v_{0,x}\tau_{rb,coll} + P_{0,x} - \frac{L}{2} + R = 0 \tag{29}$$

  To find $\tau_{rb,coll}$ for the left rail, $P_x$ must be replaced by $-\frac{L}{2} + R$.

$\tau_{rb,coll}$ for the upside rail and downside rail can be found by using the $y$ component of the displacement equation (4).

$$P_y = P_{0,y} + v_{0,y}t - \frac{1}{2}\mu_s g t^2 \widehat{u}_y \tag{30}$$

Then, the ball collides with the upside rail if $P_y = \frac{l}{2} - R$ and it collides with the downside rail if $P_y = -\frac{l}{2} + R$. the time $\tau_{rb,coll}$ is found by solving the quadratic equation (31) and taking the smallest positive solution:

$$-\frac{1}{2}\mu_s g \tau_{rb,coll}^2 \widehat{u}_y + v_{0,y}\tau_{rb,coll} + P_{0,y} - \frac{l}{2} + R = 0 \tag{31}$$

The time for the collision with the downside rail is found by analogy.

- **The ball is in rolling state**

Using the displacement equation for a ball in rolling state (7),the time $\tau_{rb,coll}$ can be found. To detect the collision with the right or the left rail, the $x$ component of equation (7) is used.

$$P_x = P_{0,x} + v_{0,x}t - \frac{5}{14}\mu_r g t^2 \widehat{u}_x \tag{32}$$

The ball collides with the right rail if $P_x = \frac{L}{2} - R$ since $P_x$ is the position of the ball center. Hence, the time it collides with the right rail $\tau_{rb,coll}$ can be found by solving the quadratic equation (33) and taking the smallest positive solution:

$$-\frac{5}{14}\mu_r g \tau_{rb,coll}^2 \widehat{u}_x + v_{0,x}\tau_{rb,coll} + P_{0,x} - \frac{L}{2} + R = 0 \tag{33}$$

To find $\tau_{rb,coll}$ for the left rail, $P_x$ must be replaced by $-\frac{L}{2} + R$.

$$-\frac{5}{14}\mu_r g \tau_{rb,coll}^2 \widehat{u}_x + v_{0,x}\tau_{rb,coll} + P_{0,x} + \frac{L}{2} - R = 0 \tag{34}$$

The time $\tau_{rb,coll}$ for the upside rail and downside rail can be found by using the $y$ component of the displacement equation (7).

$$P_y = P_{0,y} + v_{0,y}t - \frac{5}{14}\mu_r g t^2 \widehat{u}_y \tag{35}$$

Then, the ball collides with the upside rail if $P_y = \frac{l}{2} - R$ and it collides with the downside rail if $P_y = -\frac{l}{2} + R$. the time $\tau_{rb,coll}$ for the upside rail is found by solving the quadratic equation (36) and taking the smallest positive solution:

$$-\frac{5}{14}\mu_r g\tau_{rb,coll}^2\widehat{u}_y + v_{0,y}\tau_{rb,coll} + P_{0,y} - \frac{l}{2} + R = 0 \tag{36}$$

The time for the collision with the downside rail is found by analogy :

$$-\frac{5}{14}\mu_r g\tau_{rb,coll}^2\widehat{u}_y + v_{0,y}\tau_{rb,coll} + P_{0,y} + \frac{l}{2} - R = 0 \tag{37}$$

### 3.3.6 Ball-ball collision event

The methodology to find the time $\tau_{bb,coll}$ when one ball collides with another one is similar as previously. However, the ball is not at a fixed position like the rail. Hence, there are several possibilities:

- One ball is in sliding state and the other one is in sliding state.
- One ball is in sliding state and the other one is also in rolling state.
- One ball is in sliding state and the other one is at rest.
- One ball is in rolling state and the other one is also in rolling state.
- One ball is in rolling state and the other one is at rest.

Using (4) and (7), a general displacement equation (38) can be written:

$$\mathbf{P} = \mathbf{P_0} + \mathbf{v_0}t - \mathbf{k}t^2 \tag{38}$$

where $\mathbf{k} = \frac{1}{2}\mu_s g\hat{\mathbf{u}}$ if the ball is in sliding state or $\mathbf{k} = \frac{5}{14}\mu_r g\hat{\mathbf{v}}$ if the ball is in rolling state. If one ball is at rest, its displacement equation is reduced to $\mathbf{P} = \mathbf{P_0}$ and the solution is similar to a ball-rail collision, where the rail position is replaced by the ball position.

The equation for the first ball is indexed with $b_1$ and the second is indexed with $b_2$. A ball collides with another when $\|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\| = 2R$. Since, each side of the equation is positive, one can write $\|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\|^2 = 4R^2$ . The left-hand side of the equation can be written as follows:

$$\begin{aligned}\|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\|^2 &= \left[\sqrt{(P_{b_1,x} - P_{b_2,x})^2 + (P_{b_1,y} - P_{b_2,y})^2}\right]^2 \\ &= (P_{b_1,x} - P_{b_2,x})^2 + (P_{b_1,y} - P_{b_2,y})^2 \\ &= P_{b_1,x}^2 + P_{b_2,x}^2 - 2P_{b_1,x}P_{b_2,x} + P_{b_1,y}^2 + P_{b_2,y}^2 - 2P_{b_1,y}P_{b_2,y}\end{aligned}$$

Then, replacing each term by adequate $x$ or $y$ component of equation (38):

$$\|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\|^2 = at^4 + bt^3 + ct^2 + dt + e \tag{39}$$

where:

$$\begin{aligned}
a &= g^2\left[(k_{x,b_1} - k_{x,b_2})^2 + (k_{y,b_1} - k_{y,b_2})^2\right] \\
b &= -2g\left[(v_{x,b_1} - v_{x,b_2})(k_{x,b_1} - k_{x,b_2}) + (v_{y,b_1} - v_{y,b_2})(k_{y,b_1} - k_{y,b_2})\right] \\
c &= (v_{x,b_1} - v_{x,b_2})^2 + (v_{y,b_1} - v_{y,b_2})^2 - 2g[(P_{x,b_1} - P_{x,b_2})(k_{x,b_1} - k_{x,b_2}) \\
&\quad + (P_{y,b_1} - P_{y,b_2})(k_{y,b_1} - k_{y,b_2})] \\
d &= 2(P_{x,b_1} - P_{x,b_2})(v_{x,b_1} - v_{x,b_2}) + 2(P_{y,b_1} - P_{y,b_2})(v_{y,b_1} - v_{y,b_2}) \\
e &= (P_{x,b_1} - P_{x,b_2})^2 + (P_{y,b_1} - P_{y,b_2})^2
\end{aligned}$$

The time $\tau_{bb,coll}$ the two balls collides can finally be found by solving (40) and taking the smallest real and positive solution:

$$a\tau_{bb,coll}^4 + b\tau_{bb,coll}^3 + c\tau_{bb,coll}^2 + d\tau_{bb,coll} + e - 4R^2 = 0 \tag{40}$$

## 3.4 Simulation Algorithm

In this section, the simulation is described from the starting cue strike until the 3 balls are stationary. Between theses two moments, several events can occur. The cue strike starting event is described directly by the five parameters $a, b, \phi, \theta$ and $V$. This strike is translated into the ball initial linear speed and rotational speed using (15) and (16).

Once the velocities are calculated, the simulator will search directly for the next event. To do this, for each of the 3 balls, it calculates the time $\tau$ for each event quoted in the previous section. Now that all next events time and type are known and stored, the simulator isolates the event which has the smallest positive time to happen.

The next event type and time $\tau$ being now determined, the balls moves using their motion type (sliding, rolling or stationary) until the next event at $t = \tau$.

An example case can be seen in figure 13 and 14. This algorithm repeats until the three ball states are all stationary.

**Algorithm 1** Simulation algorithm

Initialize the ball set $\mathcal{B}$ =$(b_1, b_2, b_3)$.
Initialize an empty set $\mathcal{T}$.
Initialize cue strike with input parameters : $a, b, \phi, \theta$ and $V$.
Transform these parameters into initial impulsion $\mathbf{v_0}$ and $\boldsymbol{\omega_0}$.
Set the struck ball initial speeds to $(\mathbf{v_0}, \boldsymbol{\omega_0})$ and its state to sliding.
**while** $\mathcal{B}$ contains non-stationary balls **do**
  **for** $b$ in $\mathcal{B}$ **do**
    Calculate the next event $\tau_b$ for ball $b$ using the solver.
    $\tau_b \in \mathcal{T}$
  **end for**
  The next event happens at $t = \min \mathcal{T}$.
  Simulate until $t$ using each ball motion type (sliding, rolling, stationary).
  $\mathcal{T} \leftarrow \emptyset$
**end while**

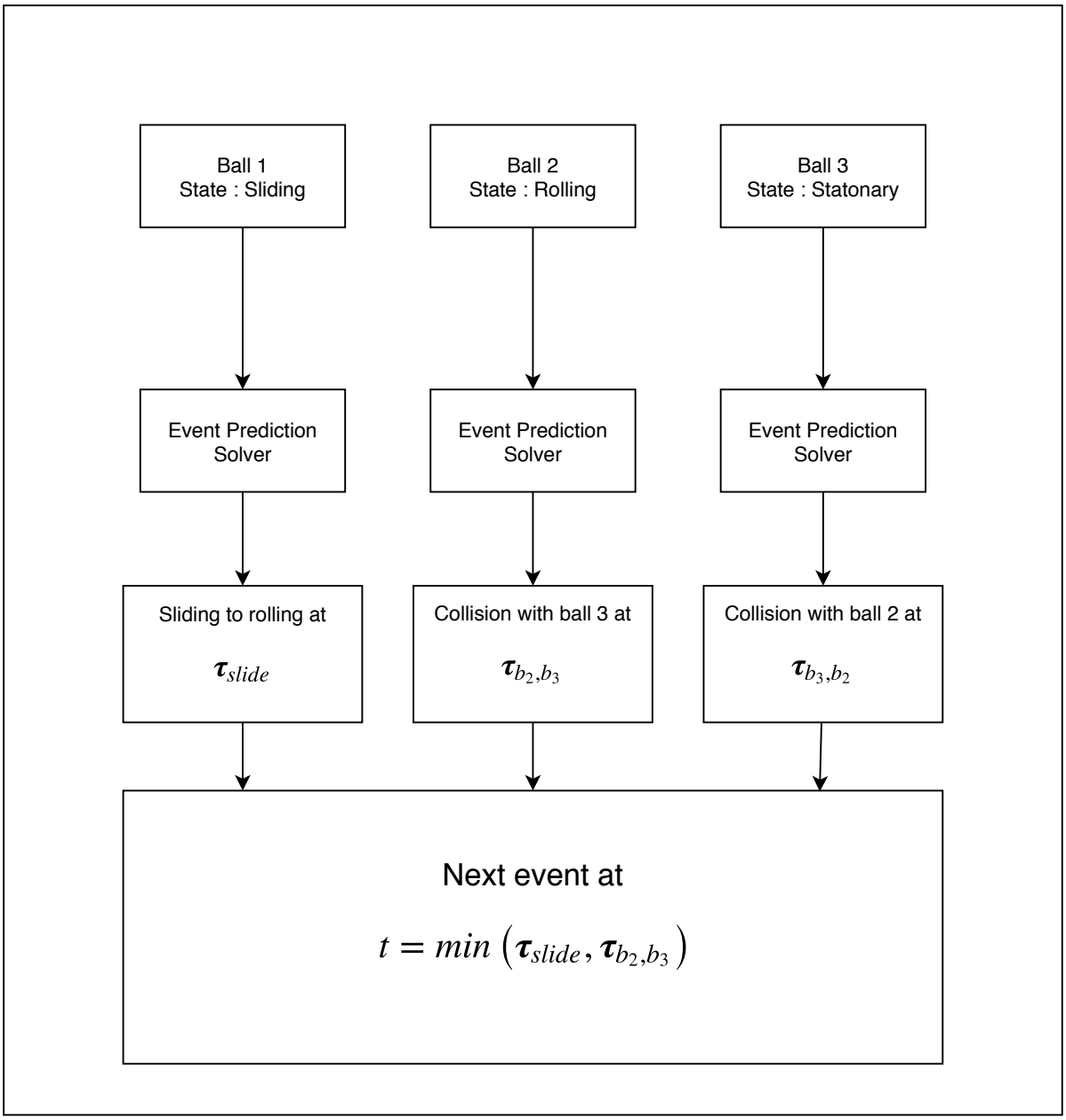



Figure 13: Example case of the general algorithm.

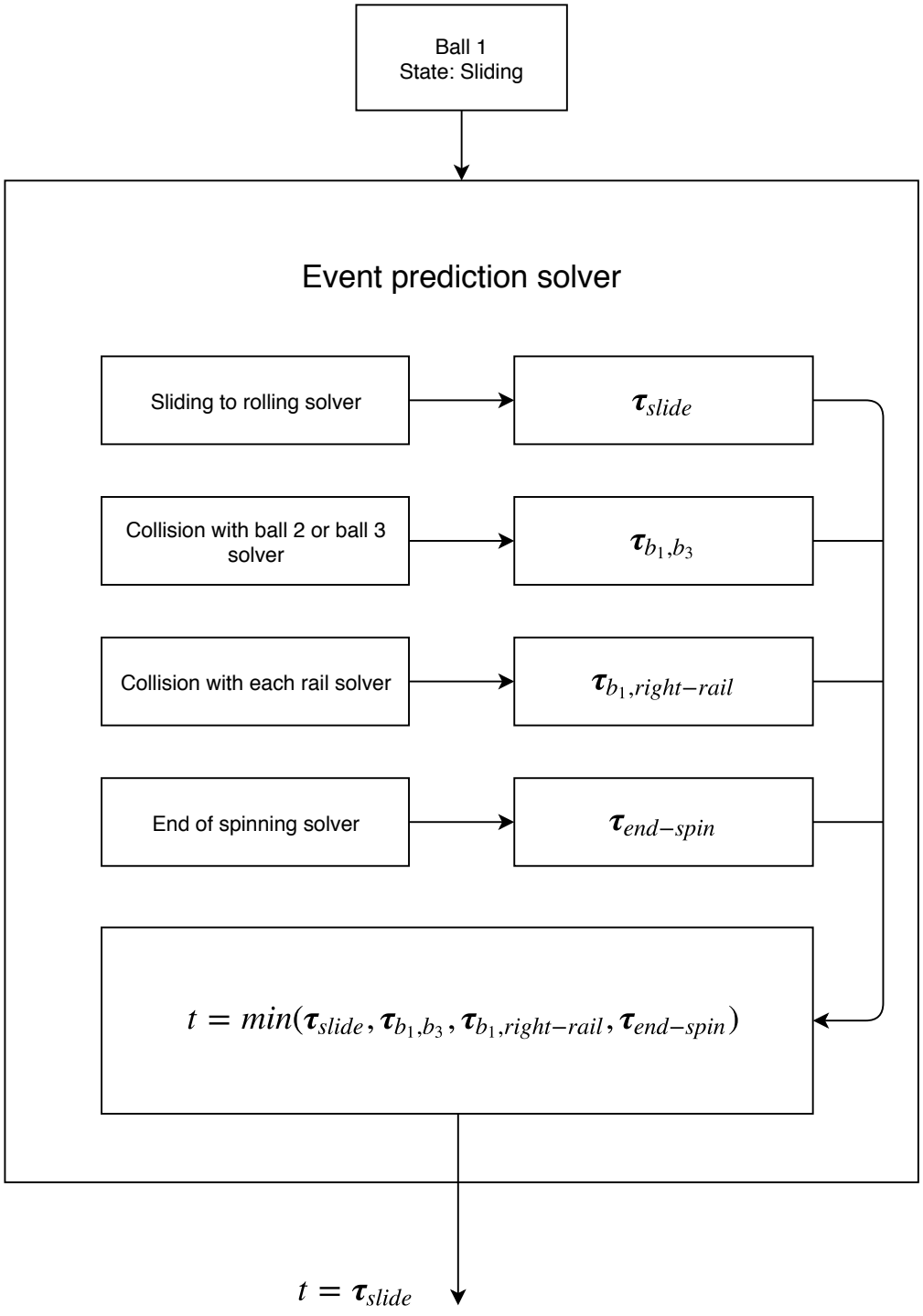



Figure 14: Example case of the event prediction solver.

# 4 Reinforcement learning

In artificial intelligence, reinforcement learning is a part of machine learning. It consists, for an agent, in learning, from experiences, the actions to be taken in a particular state, in order to optimize a quantitative reward over time. The agent acts in an environment, and makes decisions based on his current state. In return, the environment provides the agent with a reward and a next state. This reward can be high if the agent took a good action in this particular state or it can be low if the action was bad.

Through iterative experiences, the agent seeks an optimal decision-making behaviour (called optimal policy), which is a function associating the action to be performed with a each particular state, in the sense that it maximizes the sum of the rewards over time. In machine learning, the environment is typically formulated as a Markov Decision Process (MDP).

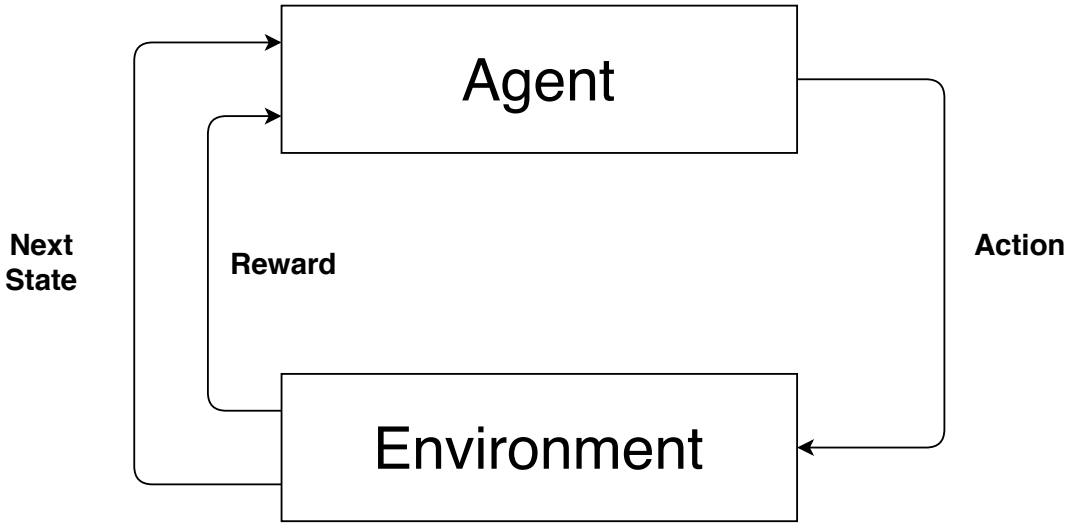



Figure 15: Agent-Environment interaction in Reinforcement learning.

## 4.1 Decision-Making Problem Formulation

### 4.1.1 Markov Decision Process

The problem formulation is in the form of a Markov decision process. MDPs provide a mathematical model to simulate decision making problem, they can

be helpful for studying optimization problems solved with reinforcement learning. The problem must satisfy the following Markov property. A state $s_t$ from the environment satisfies the Markov property if and only if :

$$P(s_{t+1} \mid s_t) = P(s_{t+1} \mid s_1, ..., s_{t-1}, s_t) \tag{41}$$

This means the future is only determined from the present and not the past. This equation is discussed in the section 4.1.2 where the states of the problem are defined.

### 4.1.2 State description

The states space set $\mathcal{S}$ of the problem is made of the state set of each ball $b_1, b_2$ and $b_3$. Each ball state set consists in two continuous variables, where one represents the $x$ ball coordinate and the other represents the $y$ ball coordinate.

$$\mathcal{S} = \mathcal{S}_{b_1} \times \mathcal{S}_{b_2} \times \mathcal{S}_{b_3} \tag{42}$$

A particular state of the set $s_t \in \mathcal{S}$ is:

$$s_t = (P_{b_1,x}, P_{b_1,y}, P_{b_2,x}, P_{b_2,y}, P_{b_3,x}, P_{b_3,y}) \tag{43}$$

That means all carom games configurations can be represented by those six variables which completely characterizes a state of the game.

There are some physical constraints that are applied to the state variables since the position is limited by the rails around the table surface. Furthermore, two or

more balls cannot be at the same place.

$$
\begin{aligned}
P_{b_1,x} &\in \left[\frac{-L}{2}+R, \frac{L}{2}-R\right] \\
P_{b_2,x} &\in \left[\frac{-L}{2}+R, \frac{L}{2}-R\right] \\
P_{b_3,x} &\in \left[\frac{-L}{2}+R, \frac{L}{2}-R\right] \\
P_{b_1,y} &\in \left[\frac{-l}{2}+R, \frac{l}{2}-R\right] \\
P_{b_2,y} &\in \left[\frac{-l}{2}+R, \frac{l}{2}-R\right] \\
P_{b_3,y} &\in \left[\frac{-l}{2}+R, \frac{l}{2}-R\right] \\
\|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\| &\geq 2R \\
\|\mathbf{P_{b_1}} - \mathbf{P_{b_3}}\| &\geq 2R \\
\|\mathbf{P_{b_2}} - \mathbf{P_{b_3}}\| &\geq 2R
\end{aligned}
$$

Hence, this definition of a state shows it respects the Markov property (41). Indeed, next state (the position of each ball after a move) is completely characterized by the current state (the position of each ball before the move) and the shot parameters. And so, the past states and actions (called history) are entirely described by the present state.

### 4.1.3 Action description

The player shot is modeled through the set of actions $\mathcal{A}$. An action $a_t$ in the set $\mathcal{A}$ models of the continuous cue input parameters.

$$a_t = (\alpha, \beta, \phi, \theta, V) \tag{44}$$

where $\alpha = \dfrac{a}{R}$ and $\beta = \dfrac{b}{R}$ and the other parameters are described in the cue-ball collision section.

Some physical constraints are applied to these variables. The cue strike must touch the ball.

$$a^2 + b^2 \leq R^2 \tag{45}$$

This leads to:

$$\alpha^2 + \beta^2 \leq 1 \tag{46}$$

Sometimes, a ball or a rail can block particular action, a simplifying assumption is made here and theses shots are allowed. As can be seen in figure 16 and 17, the shot cannot be performed if the angle $\theta$ decreases.

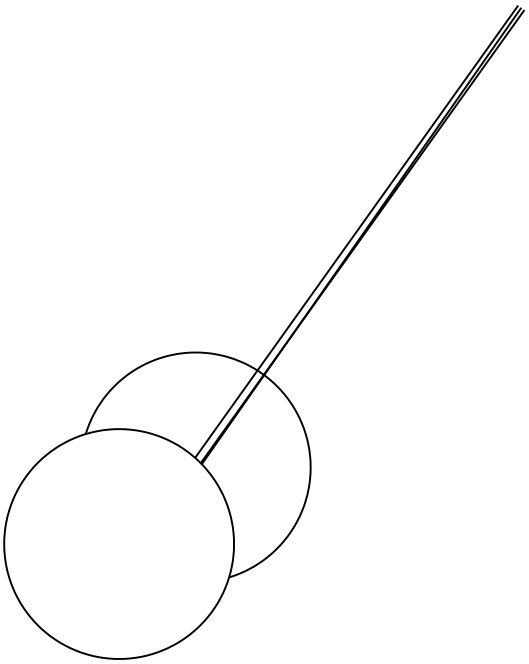


Figure 16: A ball blocking a particular shot.

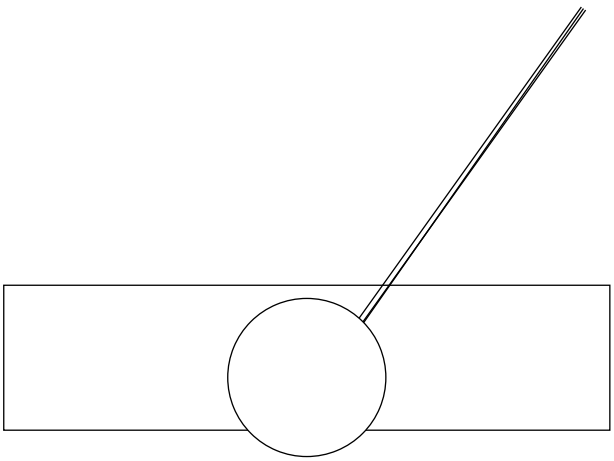


Figure 17: A rail blocking a particular shot.

In order to reduce the actions space size i.e. to simplify the search of an optimal policy, the parameter $\theta$ is set to 10° and the parameters $\alpha$ and $\beta$ are both set to 0. Hence, the remaining continuous action variables are the cue speed at the impact $V$ and the aiming angle direction $\phi$. The maximum speed $V$ is set to 6 [m/s].

### 4.1.4 Transition function

The transition function $f$ from a state $s_t$ to a state $s_{t+1}$ is totally described by the physics of the system. The state $s_{t+1}$ depends only on the previous state $s_t$, the action $a_t$ and the physics involved in the simulator. The evolution of the system is described through the relation:

$$s_{t+1} = f\left(s_t, a_t\right) \tag{47}$$

A player cannot apply the precise actions of a recommendation system since he does not have the exact information of the shots he is playing but only an approximation. A transition function to model this phenomenon would take into account any noise added to the action. However, for this work this approach has been simplified to have a deterministic transition function.

### 4.1.5 Reward function

Several reward functions $r$ have been designed. They are designed to achieve increasingly complex objectives. The first reward function is designed to reach a first goal: choose an action that leads at least to one collision.

$$r_1 = \begin{cases} 1, & \text{if one (or more) collision is detected} \\ 0, & \text{otherwise} \end{cases} \tag{48}$$

The collision detection implementation is really facilitated by the event-based model.

Some variants of this reward function are designed in order to take into account the shot difficulty. In this case, if a difficult strike leads to a collision, the reward is equal to 0. The strike difficulty function is defined in a later section.

A shaped reward function $r_2$ is designed in order to help the system to find solutions. In general for the carom billiards game, the player estimates a table state which has balls closer to each other better than a table where the balls are far apart from each others as it is in general easier to strike other balls. To do this, a distance reward term $r_d$ is added to $r_1$. The total distance can be written as follows:

$$d_{tot} = \|\mathbf{P_{b_1}} - \mathbf{P_{b_2}}\| + \|\mathbf{P_{b_1}} - \mathbf{P_{b_3}}\| + \|\mathbf{P_{b_2}} - \mathbf{P_{b_3}}\| \tag{49}$$

And the distance $r_d$ reward is :

$$r_d = 1 - \frac{d_{tot}}{\max\limits_{\mathbf{P_{b_1}},\mathbf{P_{b_1}},\mathbf{P_{b_1}}} d_{tot}} \tag{50}$$

Defining the distance reward in this way, a distance reward of approximately 1 means the balls are very close and a distance reward of 0 means the balls are far apart from each other. Thus, $r_2$ is defined as follows:

$$r_2 = \begin{cases} 1 + r_d, & \text{if one (or more) collision is detected} \\ r_d, & \text{otherwise} \end{cases} \tag{51}$$

Once a model is validated with reward function $r_1$ or $r_2$, it is then tested with $r_3$ or $r_4$. Reward functions $r_3$ is the boolean reward function similar to $r_1$ but for two balls collisions.

$$r_3 = \begin{cases} 1, & \text{if the cue ball collides the two other balls} \\ 0, & \text{otherwise} \end{cases} \tag{52}$$

And $r_4$ is the shaped reward function similar to $r_2$ but for two balls collisions.

$$r_4 = \begin{cases} 1 + r_d, & \text{if the cue ball collides the two other balls} \\ r_d, & \text{otherwise} \end{cases} \tag{53}$$

## 4.2 Q-learning

Q-learning [14][15] is a reinforcement learning technique. This technique is designed for discrete actions and states spaces. The agent policy is represented by a Q-Table where the rows are the different possible states and the columns are the different possible actions. Each state-action pair is characterized by a Q-value at the row of the state and at the column of the action. This Q-value represents an estimation of the expected sum of discounted rewards by taking the action represented by the column in the state represented by the row. Hence, the optimal action to take, for an agent in a particular state, is the action with the maximum q-value. In reinforcement learning, an episode is the set of the tuples $(s_t, a_t, r_t, s_{t+1})$ from the starting state $s_0$ to a terminating state of the MDP.

The goal of Q-learning, after a certain number of episodes, is to find, starting from a Q-table initialized to 0, a table that leads to the optimal policy. The Q-learning is a model-free algorithm, the agent has to explore the MDP in order

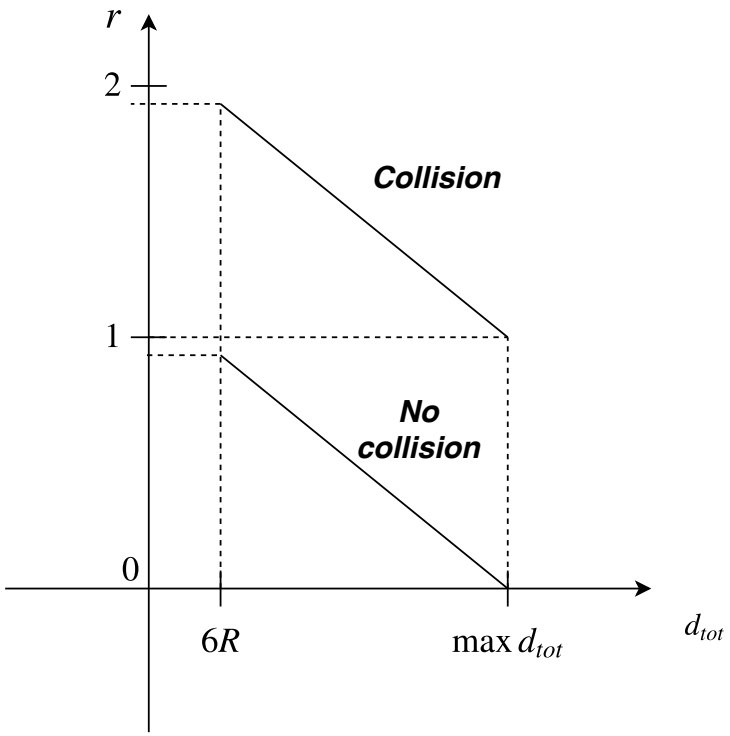



Figure 18: Reward shaping for $r_2$ and $r_4$.

to have information about its environment. This algorithm is part of Temporal difference Off-policy algorithms for control. Contrary to Monte-Carlo learning that needs a full episode to update the policy, Q-learning is online. That means Q-table (from which the policy is derived) is updated after each action taken from the agent. The Q-table update is done through the Bellman equation:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)\, Q(s_t, a_t) + \alpha \left( r_t + \gamma \max_a Q(s_{t+1}, a) \right) \tag{54}$$

where

- $\alpha$ is the learning rate.
- $\gamma$ is the discount factor.

### 4.2.1 Exploration-Exploitation dilemma

In a given state, the agent chooses the action with the highest quality (exploitation). However, sometimes the agent must do some random exploration of the

environment to see if there are some opportunities to have higher rewards. This is called the exploration-exploitation dilemma. To allow some exploration to agent, an $\epsilon$-greedy approach is used.

This approach says the agent takes a random action (exploration) with a probability of $\epsilon$ and the agent takes the maximum quality action with a probability $1 - \epsilon$. The $\epsilon$ function is tuned in such a way that the agent explores a lot in the earlier episodes and exploit a lot in the latest episodes. As a result the agent search a lot high rewards in the environment and after a good exploration, it starts to exploit the good states and the good actions it founds when exploring.

### 4.2.2 Application to Carom

As said previously, the Q-learning is a technique made for a discrete actions and states space to obtain Q-table representation. Since the carom states space and actions space are continuous, they have been discretized in order to apply Q-learning. The discretization is done as follows.

#### Actions space discretization

Considering the input actions $a = 0, b = 0$ and $\theta = 10°$ to be constant, only $\phi$ and $V$ are discretized.

- $\phi$ is discretized from $-180°$ to $180°$ with a step of $5°$.
- $V$ is discretized from 0.5 [m/s] to 6[m/s] with a step of 0.5 [m/s].

The number of possible actions $n_{actions}$ i.e. the number of couples ($\phi$, $V$) is $73 \times 12 = 876$

#### States space discretization

In order to discretize the state space, the pool table is cut into small squares as can be seen in figure 20. Hence, using this discretization, the state is defined by 3 squares which are the ones nearest to the center of each ball.

The states space discretization is quite problematic, it brings many disadvantages. First, two tables with slightly different ball positions can represents a same state (same squares). While having the same state, a high-reward action in one configuration could result in a low-reward one in the other one. To solve this problem, the discretization can be done with smaller squares. But doing this way leads

to the high number of states. The number of states can be calculated as follows:

$$n_{states} = \left(n_{rows} \times n_{col}\right)^3 \tag{55}$$

Using a coarse discretization by setting for example $n_{rows} = 6$ and $n_{col} = 12$ leads causes the number of states to be 373248. A smaller and refined discretization leads to a number of states extremely large. This problem cannot be handled by a Q-learning technique since the Q-table size would be too large.

Some tests have been done with coarse discretization, but as expected, the results after 50000 episodes, even for one collision reward, are as good as a totally random policy.

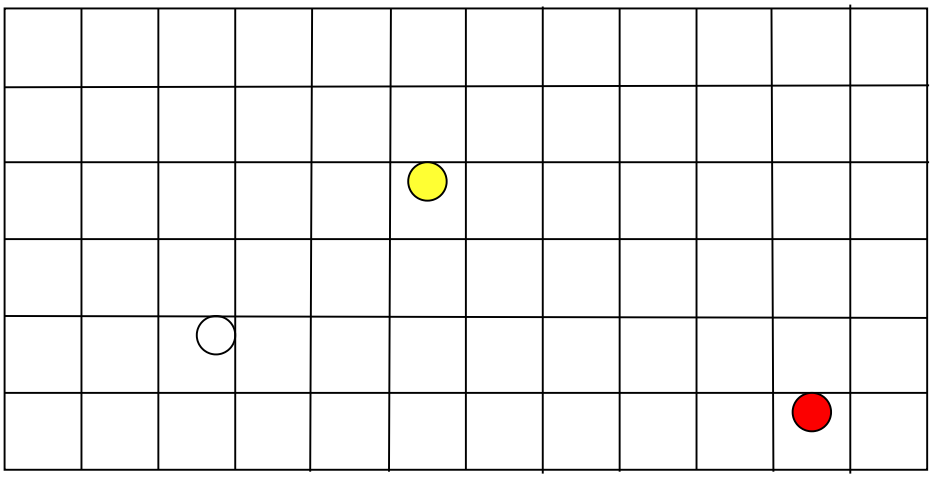


Figure 19: States space discretization.

**Algorithm 2** Q-learning algorithm

Initialize $Q(n_{states}, n_{actions})$ arbitrarily.
Observe initial state $s$ of the environment.
**repeat**
  Select action $a$ using $\epsilon$-greedy approach
  Carry out the action $a$
  Observe reward $r$ and the next state from the environment
  Update $Q(s, a)$ using Bellman equation
  Set the current state to the next state
**until** last episode

## 4.3 Deep-Q-learning

In order to deal with the continuous states space and to avoid states space discretization, the use of functions approximators are mandatory for this problem. Hence, there is no more Q-table and the scalability problem is solved by using a neural network to approximate the $Q$ function [16].

The neural network has as input the states space and it has as output a Q-value for each action. Instead of a Q-table update after each action, now the weights of network $\theta_i$ are updated. The optimal policy represented before as an optimal Q-table $Q^*$, is now given through optimal weights $\theta_i$:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{56}$$

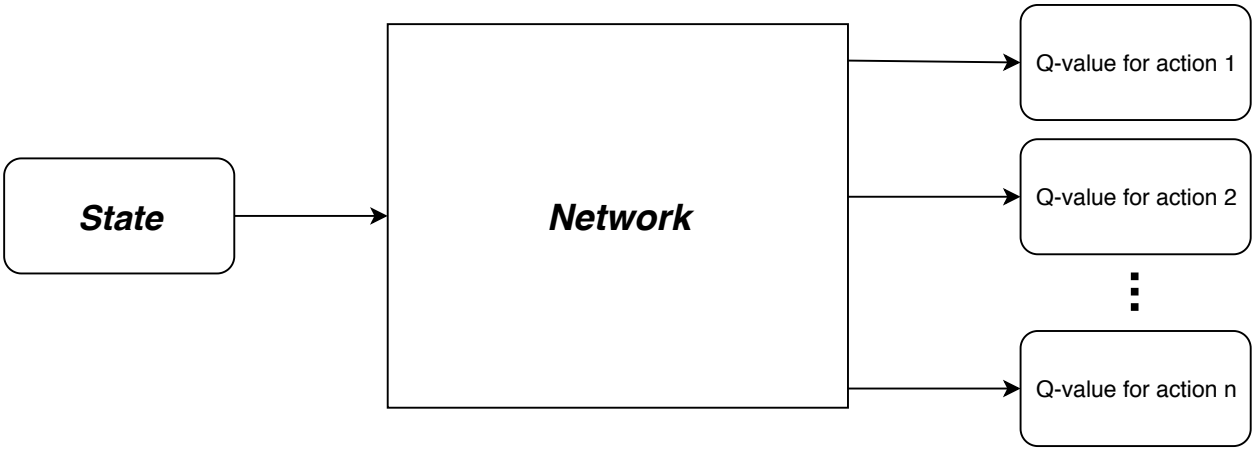



Figure 20: Deep-Q-Network.

The neural network weights are updated using backpropagation with a loss function, this loss function $L(\theta_i)$ represents the mean-squared error in the Bellman equation where optimal target values $r_t + \gamma \max_a Q^*(s_{t+1}, a_{t+1})$ are replaced by approximate target values $r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta_i^-)$ where $\theta_i^-$ are the parameters of the network at a previous iteration.

$$L = \mathbb{E}\left[\left(\underbrace{r_t + \gamma \max_a Q^*(s_{t+1}, a_{t+1}; \theta_i^-)}_{target} - Q(s_{t+1}, a_{t+1}; \theta_i)\right)^2\right] \tag{57}$$

The neural network weights update is done as:

- Given the current state, a feedforward pass is done to get Q-values for the actions.
- Given the next state, a feedforward pass is done and the maximum Q-value action output is calculated.
- The target $(r + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}; \theta_i^-))$ for the taken action is calculated using the reward and the value calculated at the previous step.
- The loss function is computed using all previous steps.
- The weights are updated using backpropagation (minimizing the loss function).

### 4.3.1 Experience Replay

As the game is played through simulation, sets containing the current state, the taken action, the reward and the next state $(s_t, a_t, r_t, s_{t+1})$ are stored in a library of experiences also called replay memory.

The neural network is not training directly from the current game simulation, but instead, the training is performed by randomly sampling mini-batches from the past simulations. The network does not train anymore on a natural continuous evolution of the system, This technique helps to avoid the system to overfit a particular evolution of the game.

### 4.3.2 Fixed Target Network

When updating weights, the loss function have to be computed. To compute the loss, the target must be calculated. Since the target depends on network parameters, it changes quickly and can lead to stability issues when training. A solution to this problem is to update the target at a lower frequency.

### 4.3.3 Application to Carom

In this case, the states space of the carom environment does not need to be discretized since neural network is used as a function approximator. However, this technique still needs discrete actions space. Hence, the actions space is discretized in the same way as before. The neural network structure and parameters can be found in the Table 2 and 3.

| Layers | Input | Activation | Output |
|---|---|---|---|
| Flatten | 6 | - | 16 |
| Fully connected | 16 | ReLU | 16 |
| Fully connected | 16 | ReLU | 16 |
| Fully connected | 16 | ReLU | 16 |
| Fully connected | 16 | Linear | $n_{actions}$ |

Table 2: DQN - Neural Network structure used.

| Parameters | |
|---|---|
| learning rate | 0.001 |
| Target update frequency | 10000 |
| Replay Memory size | 50000 |
| Discount factor | 0.99 |
| Mini-batch size | 32 |

Table 3: DQN - Main parameters used for the training.

The results for the reward design $r_1$ and $r_2$ are respectively shown in figure 21 and 22. The results are better than for the Q-learning. For the reward design $r_1$, the mean reward is stabilized at 0.4. That means the agent is able to find an action where the ball collides with another one, two times out of five. Using a reward shaping function $r_2$ does not help, the results are similar, because in this case the maximum reward that can be achieved is approximately 2. Here, the agent is also able to find a good shot two times over five.

The problem is not completely solved for one ball collision reward, it is therefore useless to evaluate this technique with two balls collision reward functions $r_3$ and $r_4$. The main problem with the deep-Q-learning is the action space that must be discrete. And in the case of carom, the discretization leads to a high number of actions. Yet this type of technique is generally used for small states space size (usually about 10 discrete actions).

## 4.4 Deep deterministic policy gradient

The deep deterministic policy gradient algorithm [17][18] overcomes the problem of continuous action since it is made only for this type of actions space. This approach is closely connected to deep-Q-learning algorithm. When computing the loss function (57), finding $\max_a Q^*$ is easy for the deep-Q-learning since the number of action is limited. But finding it with continuous action space is hard. Using an optimization algorithm to find $\max_a Q^{ast}$ is really expensive in time since

**Algorithm 3** Deep-Q-learning algorithm

Initialize network $Q$ with random weights.
Initialize target network with the same random weights.
Initialize replay memory
Observe initial state $s$ of the environment.
**repeat**
  Select action $a$ using $\epsilon$-greedy approach.
  Carry out the action $a$ in the simulator and observe reward and next state.
  Store transition $(s_t, a_t, r_t, s_{t+1})$.
  Sample random mini-batch of transitions $(s_t, a_t, r_t, s_{t+1})$ from replay memory.

  Calculate target and compute loss function.
  Update network parameters using backpropagation.
  Update the network target every 10000 steps
  Set the current state to the next state.
**until** last episode

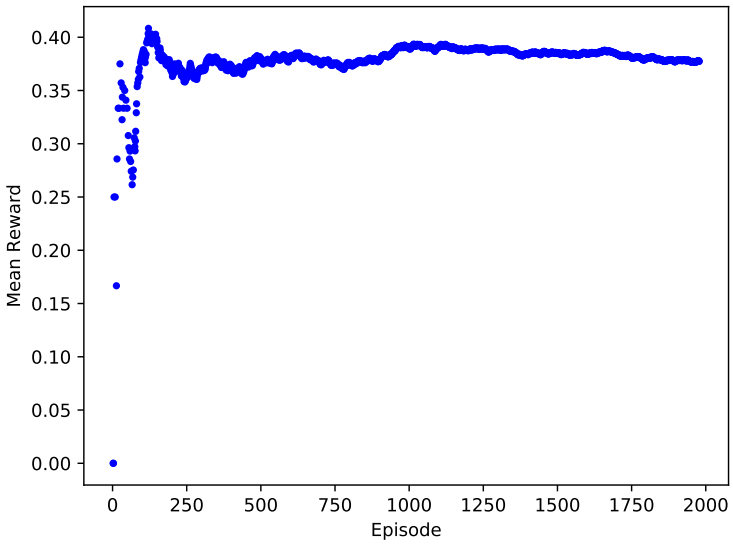



Figure 21: DQN - Mean reward using the reward design $r_1$.

the optimization problem needs to be run every time an agent take an action. The DDGP algorithm assumes $\max_a Q^*(s, a)$ to be differentiable with respect to $a$. Using this assumption allows to set up a policy $\mu(s)$ based on a gradient learning

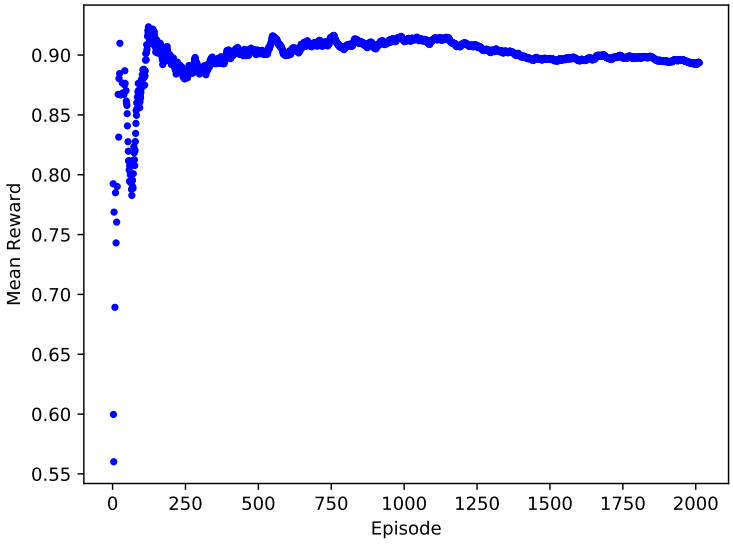



Figure 22: DQN - Mean reward using the reward design $r_2$.

rule. Hence, instead of using an optimization algorithm to solve the problem, the following approximation is made:

$$\max_a Q(s,a) \approx Q\left(s, \mu(s)\right) \tag{58}$$

The DDGP algorithm consists in 2 different types of network: the actor network which produces an action and the critic network which takes this action as input and produces action Q-value. This type of algorithm is called Actor-Critic algorithm [19]. The experience replay trick used in the Deep-Q-learning technique is also used in the DDGP algorithm.

### 4.4.1 Target Networks Update

For the DDGP algorithm, instead of updating the target networks at a low frequency, it is recommended to be updated at each step using a soft update:

$$\begin{cases} \theta_{actor,target} \leftarrow k\theta_{actor,target} + (1-k)\theta_{actor} \\ \theta_{critic,target} \leftarrow k\theta_{critic,target} + (1-k)\theta_{critic} \end{cases} \tag{59}$$

Where $k$ is a parameter between 0 and 1 (close to 1 for a soft update). Hence, the target values are slowly changing, improving the stability of learning.

### 4.4.2 Application to Carom

The actor network and the critic network structure used can be found in [20]. The main parameters of the algorithm are shown in Table 4. The results for the reward design $r_1$ are shown in figure 23, 24 and 25. The results are calculated with intervals of 10000 episodes. The values shown in the results are computed after each interval by getting the mean of the last 10000 episodes. The shaped reward design $r_2$ gives approximately the same results. The problem is solved, because after 70000 episodes the mean reward converges to 0.8 (which is clearly near the maximum of 1) while still exploring. This means one collision is detected 8 times out of 10 while training. However, the agent adopts a strategy of striking with high speed and a more or less horizontal aiming angle. This strategy consists in scanning a large part of table in order to easily collide with one ball. This strategy can be seen in figure 26.

| **Parameters** | |
|---|---|
| learning rate | 0.001 |
| $k$ | 0.999 |
| Replay Memory size | 50000 |
| Discount factor | 0.99 |
| Mini-batch size | 32 |

Table 4: DDGP - Main parameters used for the training.

In order to avoid this hard strategy, the reward $r_1$ is redesigned to take into account the shot difficulty [5]. To do this, the reward is reduced when the cue ball collides with rails before colliding with a ball. The higher the number of rails hit before the ball collision, the lower the reward is. Another parameter that can affect the shot difficulty (and consequently the reward), is the angle of collision. As can be seen in figure 27, the higher the angle of collision, the higher the shot difficulty is and the lower the reward is. However, currently, only the first component of the shot difficulty is taken into account to redesign $r_1$.

The result with the reward function $r_1$ redesigned with the shot difficulty is shown in figure 28. As expected, since the agent cannot adopt the strategy in figure 26, the learning takes more time. It enables to find a solution 8 times out of 10 after 250000 episodes while still exploring. The strategy adopted in this case is more natural, it simply tries to aim directly to another ball.

The problem being solved for one ball collision ($r_1$ and $r_2$), the same technique is applied to the two balls collision problem ($r_3$ and $r_4$). As can be seen in figure

**Algorithm 4** Deep deterministic policy gradient algorithm.

Initialize actor and critic networks with random weights.
Initialize target networks with the same random weights.
Initialize replay memory.
Observe initial state $s$ of the environment.
**repeat**
  Select action $a$ using actor network.
  Carry out the action $a$ in the simulator and observe reward and next state.
  Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay memory.
  Sample random mini-batch of transitions $(s_t, a_t, r_t, s_{t+1})$ from replay memory.

  Calculate critic target and compute loss function.
  Update critic network parameters using backpropagation by minimizing loss function.
  Update the actor network using the sampled gradient described in [17].
  Update the target networks using a soft update (57).
**until** last episode

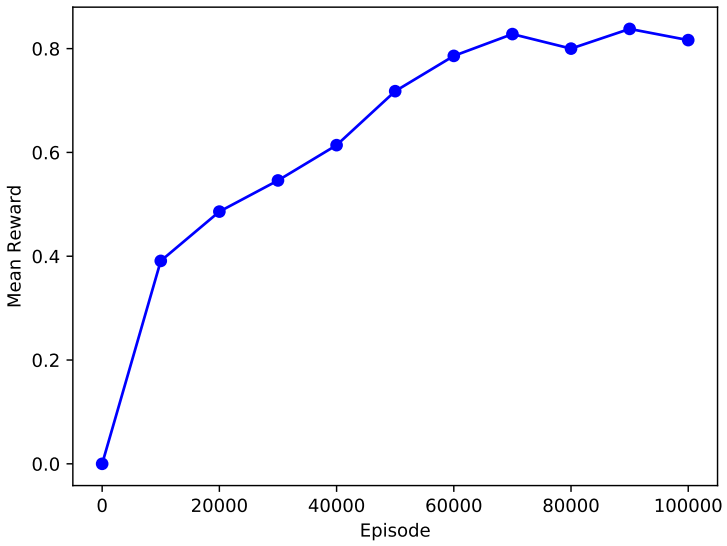



Figure 23: DDGP - Mean reward using the reward design $r_1$.

30, the reward is only able to converge to 0.05. Hence, one strike out of 20 leads to a good shot. For a two collision problem a random policy has been evaluated to

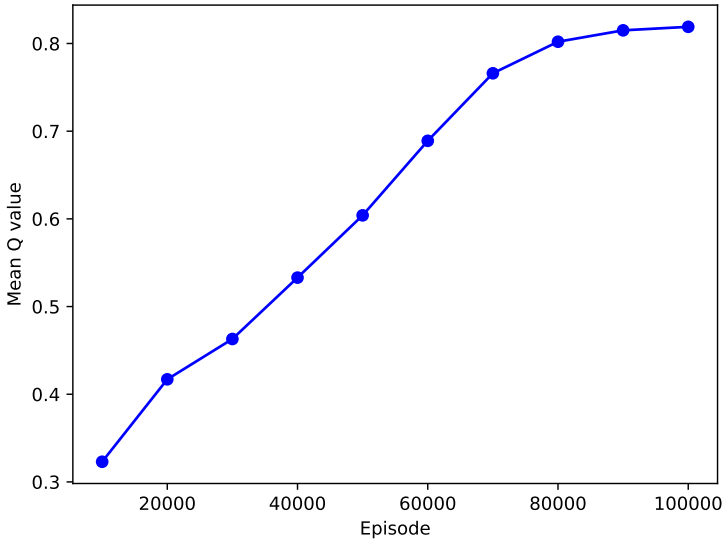



Figure 24: DDGP - Mean Q value using the reward design $r_1$.

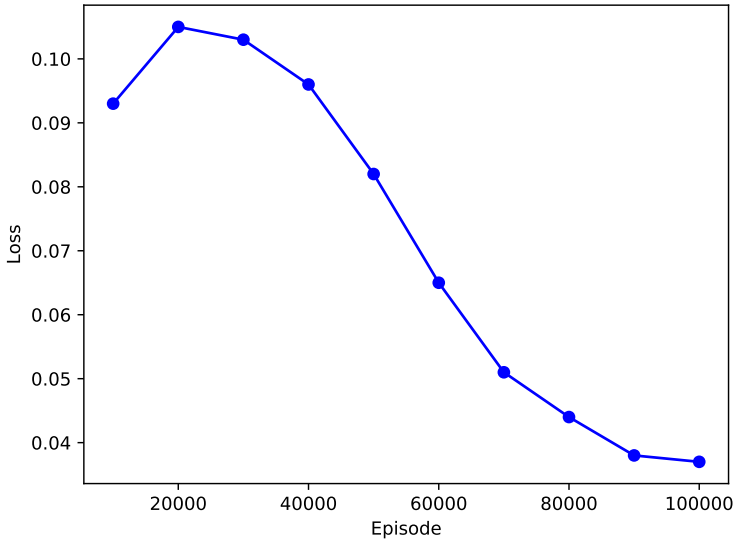



Figure 25: DDGP - Loss using the reward design $r_1$.

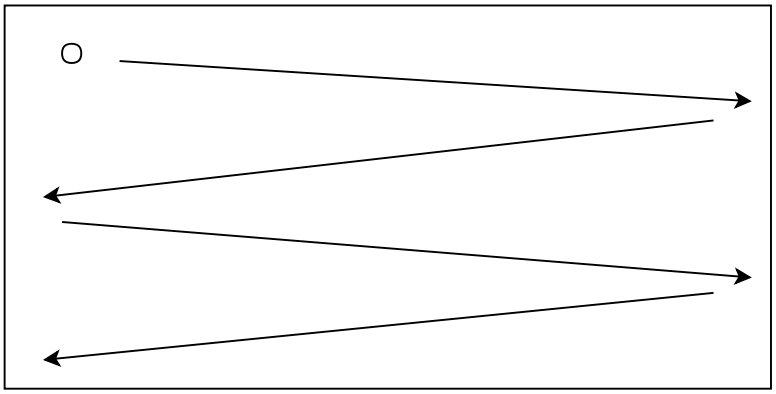


Figure 26: Strategy adopted by the agent with $r_1$.

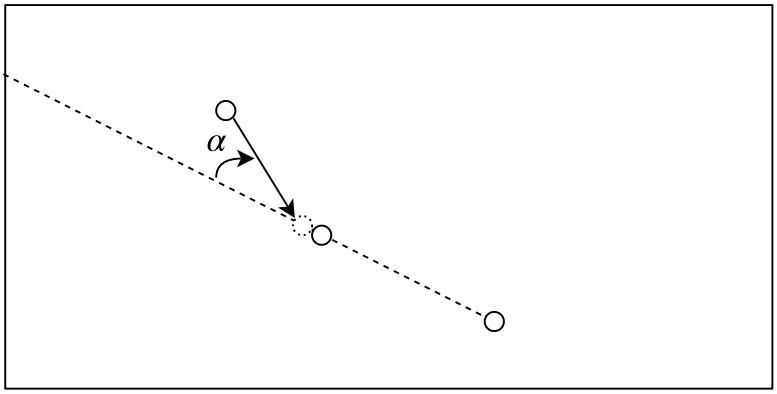



Figure 27: Shot difficulty: angle of collision parameter.

make a good strike once out of 1000. The policy found cannot solve the problem of two balls collision like the one of one ball collision, but it is still 50 times better than a random policy. For the reward design $r_4$, the policy found is bad. In this case the algorithm is only trying to make the balls closer to each other. The reward $r_4$ is overfitting the distance reward $r_d$ and forgetting the initial goal to collide with the two other balls.

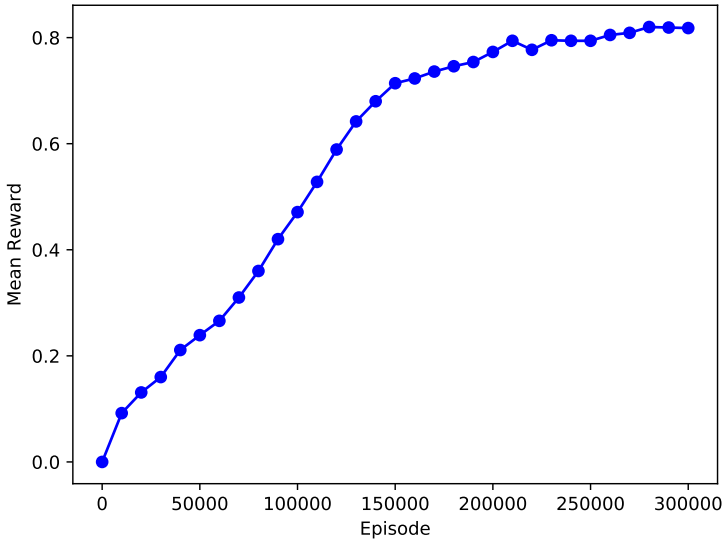



Figure 28: DDGP - Mean reward using the reward design $r_1$ taking into account the shot difficulty.

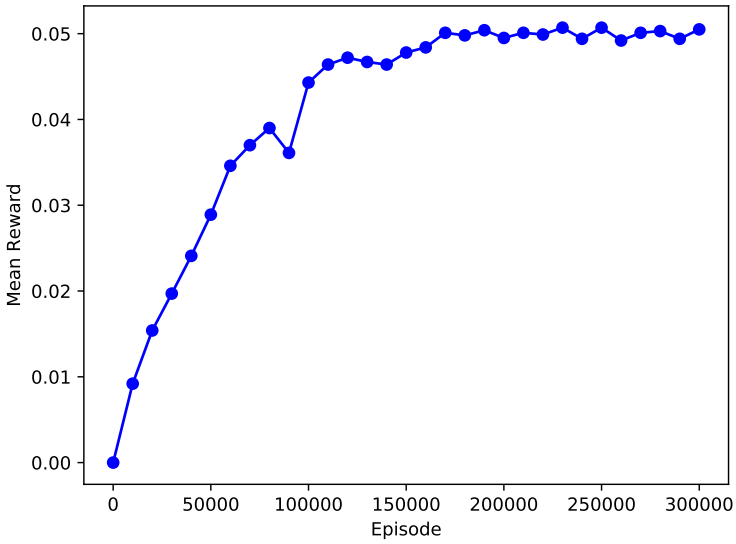



Figure 29: DDGP - Mean reward using the reward design $r_3$.

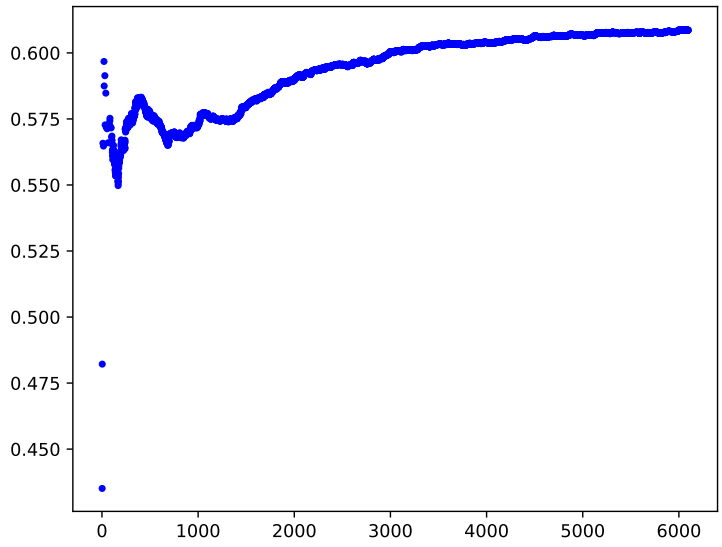



Figure 30: DDGP - Mean reward using the reward design $r_4$.

# 5 Recommender System

## 5.1 Reward Sensitivity

In science, it is stated that the same causes lead to the same effects (determinism). And in general, there is also the idea that similar causes will cause similar effects. If approximations are made at the beginning, the consequences will be limited. In a scientific experiment, everything is rarely described perfectly and small approximations can often be made without altering the final result too much. However, there are physical systems where this idea does not work [21].

For example, the simple pendulum system is not highly sensitive to initial conditions whereas the double pendulum is. That means starting with the simple pendulum from an angle very close to the initial angle will not have large consequence on the pendulum motion. However, the double pendulum system which is highly sensitive to initial state, has a completely different motion if the initial angle is modified a bit. These examples can be seen in figure 31 and 32.


Figure 31: Effect of initial conditions variation on simple pendulum.

The billiard system is totally analogous to the pendulum system. For the one ball collision system, changing a bit the position of the cue ball or the action, will result in a good reward. However, when trying to make 2 balls collision, changing a bit the position of the cue ball, leads often only to one ball collision. In this case the reward is highly sensitive to the state and the action. In figure 33, a learned state/action pair can be seen (red arrow). If a similar state (cue ball in dotted-line) is provided to the network, it produces an action similar to the one learned before (blue line). It can be seen the cue ball success to collide with the first one, but fails to collide with the second one. That is why the system learns

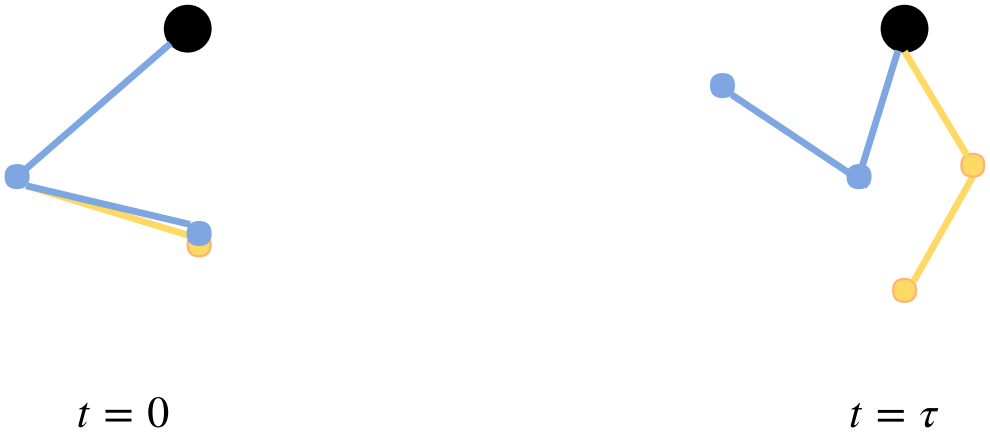



Figure 32: Effect of initial conditions variation on double pendulum.

well to collide with one ball but not with two: $r_3$ is highly sensitive to the action and to the state while $r_1$ is not.

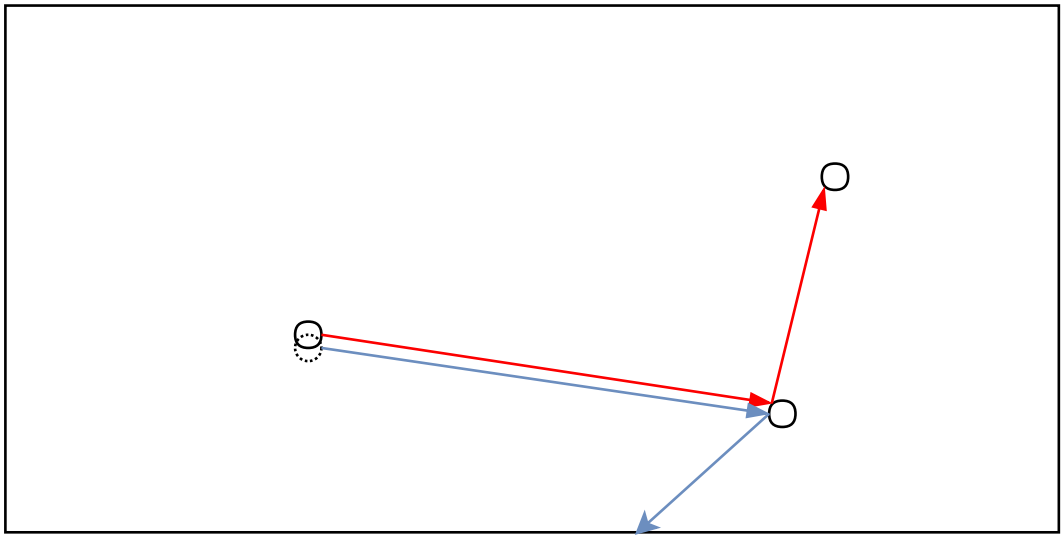


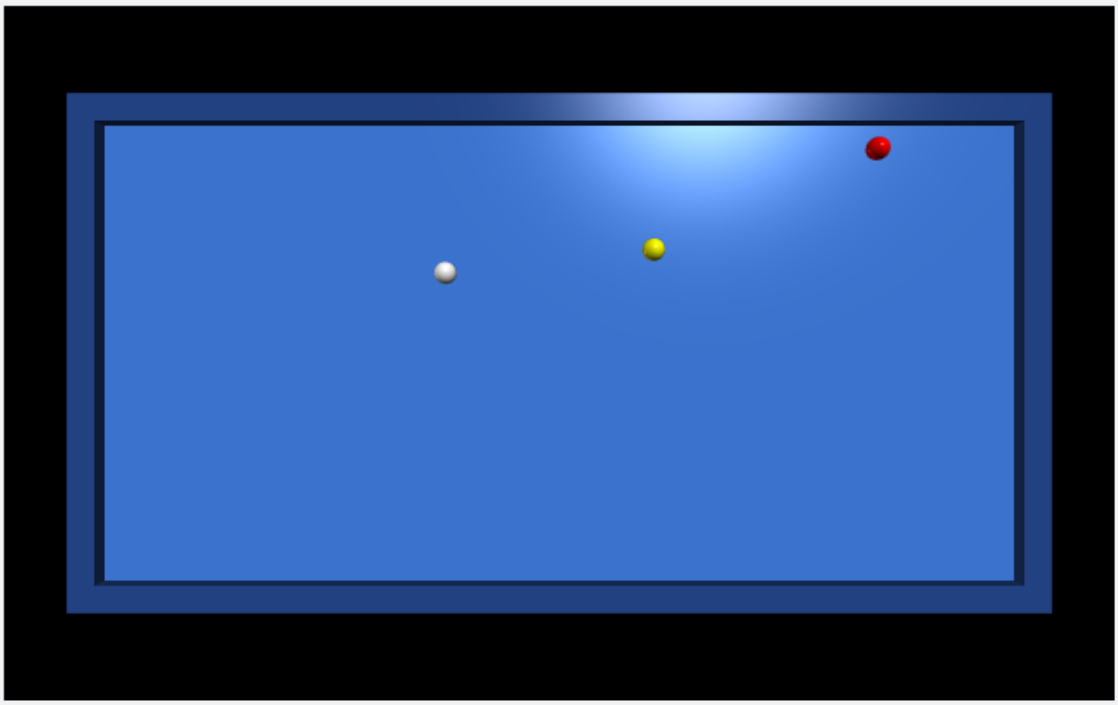
Figure 33: Reward sensitivity.

To overcome this reward sensitivity issue, the cue inputs $a$, $b$ and $\theta$ that were constant until then, are now randomly sampled until one strike that produces $r_3 = 1$ (keeping the same action variables $\phi$ and $V$ produced by the network).

## 5.2 User Interface

The recommender system is provided with a graphic interface [22] that can be seen in figure 34. When developing the simulator, this graphic interface allowed to check physical errors, to observe strategies adopted by the agent and much more things. The simulation can be done in two ways. The first way is to enable the rendering, this is a slow simulation that allows to check errors or testing the agent after training. The second one, is to only simulate the events, this fast simulation is done for training the networks.

Different parameters are provided through the interface, such as the speeds norm, the next found event, the number of episodes if the system is training and the cue inputs recommended by the simulator. In recommender system mode (figure 35), the user can adjust the position of the ball in the graphic interface by simply moving the balls with the mouse. Once this is done, the player can press the run button and the simulator will provide him successful shot parameters. And then the simulator will simulate this shot to help the player.



**LINEAR SPEED** [m/s]
WHITE: 0.198
YELLOW: 0.000
RED: 0.218

**ROTATIONAL SPEED** [deg/s]
WHITE: (-1.607,-6.277,0.000) - Norm: 6.479
YELLOW: (0.000,0.000,0.000) - Norm: 0.000
RED: (2.739,-6.617,0.000) - Norm: 7.161

**NEXT EVENT**: WHITEROL2STA

**CUE INPUTS**
a: 0.246
b: -0.441
theta: 19.850
phi: 41.937
V: 3.364

**EQUIVALENT BALL IMPULSION**
v0 = (1.685,1.514,0.000)
w0 = (81.201,-115.016,-45.605)

**EPISODE**: 1017

Figure 34: User Interface.

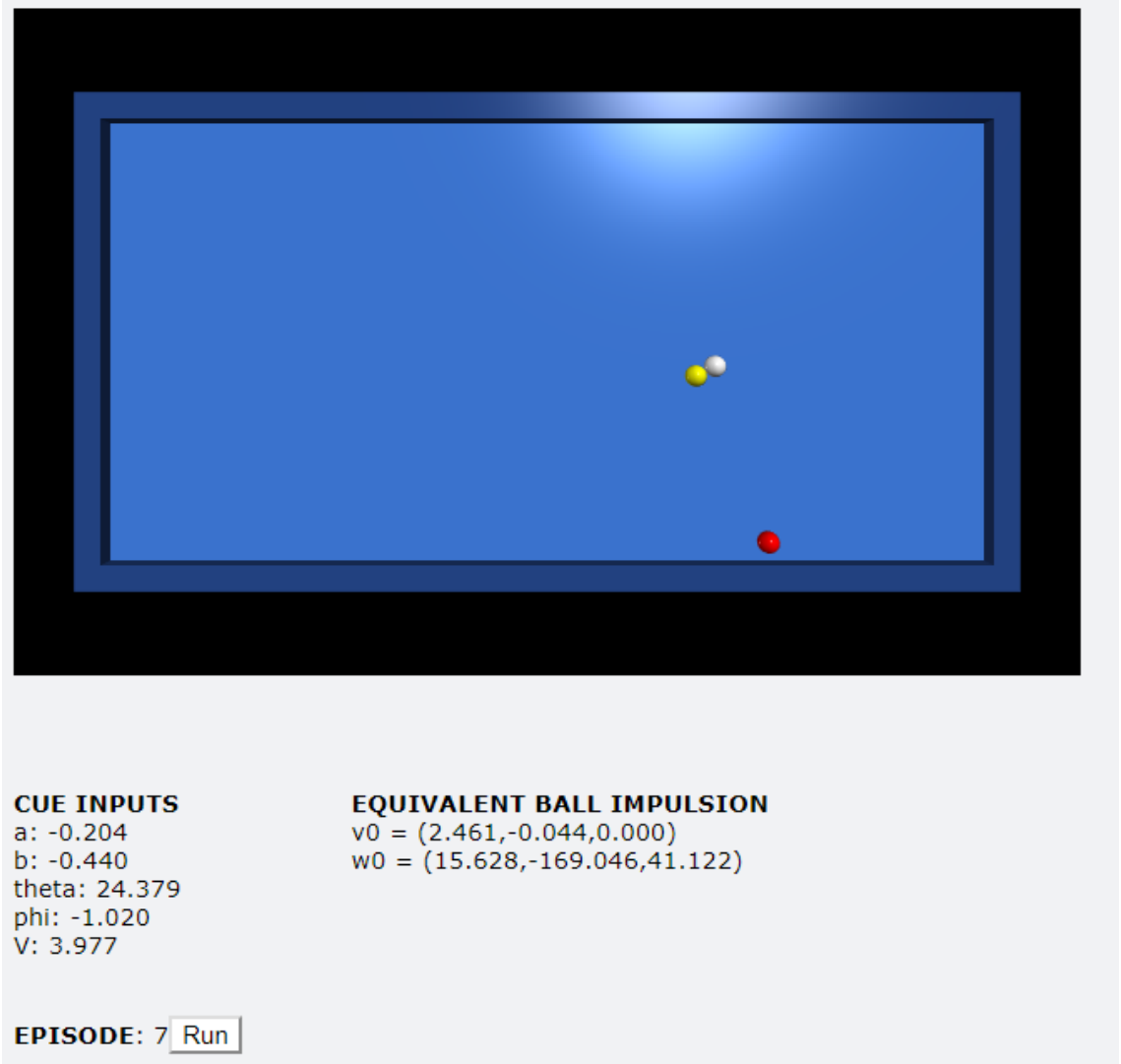



Figure 35: User Interface in recommender system mode.

# 6 Conclusion

The results of this research show the reinforcement learning approach using deep deterministic gradient algorithm is good when searching to collide with only one ball. But, due to a sparse reward, which is highly sensitive to the state of billiard table and to the taken action, the problem for two balls collision is still hard to be fully solved by only the reinforcement learning approach. Some suggestions to solve this problem can be found in the future work section. Using discretization to apply techniques such as Q-learning or Deep-Q-learning does not improve the results. However, other algorithms based on continuous actions space and states space could be tested.

The event-based simulation method applied in this work appears to work very well. Using this type of simulation makes the application of reinforcement learning much easier. Indeed when designing the reward, a collision detection implementation is mandatory. This implementation is done directly with the event-based simulator.

The recommender system is based on the output of Deep deterministic gradient policy algorithm where the actions variables $a$, $b$ and $\theta$ are modified using a noise in order to find successful shot.

## 6.1 Future Work

In the present work, the goal is solved entirely by reinforcement learning only with the reward design $r_1$. However, for the reward design $r_3$, the goal is not directly achieved by reinforcement learning, it needs a search to be done before an output is produced. Two main problems are encountered, the first one is due to high reward sensitivity as said previously, the second one is the sparse reward. Indeed, when training with $r_3$, the system finds a reward of 1 only after a very large number of episodes. In order to help the system, there exists several techniques that are presented below. Furthermore, the simulator can be changed by improving the physics.

### 6.1.1 Hindsight Experience Replay

Instead of shaping the reward function, hindsight experience replay (HER) [23] can be used only with a boolean reward (like $r_1$ or $r_3$). Sometimes, when the agent fails to receive a reward of 1, HER pretends that the agent actually has done a good action.

### 6.1.2 Learning from demonstrations

Learning from demonstrations [24][25] is an approach that provides the agent with demonstrations by an expert. In the case of billiard, it would be by filming expert games of carom and storing all expert game transitions in a data set from which the agent can learn some particular behaviour with a non-sparse reward.

It is also possible to combine computer vision and reinforcement learning techniques to achieve this goal.

### 6.1.3 Improvement of simulator physics

An assumption has been made for this work : the balls cannot be in a flying state. Hence, some equations have been modified to take into account the ball $z$ component position cannot change distancing us from realistic simulator. Then, it is possible to improve the simulator by adding a flying state. Furthermore, friction coefficients are supposed to be constant over the whole surface and rails. This can be improved by finding empirical values for these coefficients.

# References


[1] Fédération Française de Billard. Code sportif billard carambole. 2011.

[2] Gaspard-Gustave Coriolis. *Théorie mathématique des effets du jeu de billard*. Carilian-Goeury, 1835.

[3] Régis Petit. *Billard: théorie du jeu*. Chiron, 2004.

[4] Wayland C Marlow. *The physics of pocket billiards*. MAST, 1995.

[5] Michael Smith. Pickpocket: A computer billiards shark. *Artificial Intelligence*, 171(16-17):1069–1091, 2007.

[6] Jean-François Landry, Jean-Pierre Dussault, and Philippe Mahey. A robust controller for a two-layered approach applied to the game of billiards. *Entertainment Computing*, 3(3):59–70, 2012.

[7] Michael Greenspan, Joseph Lam, Marc Godard, Imran Zaidi, Sam Jordan, Will Leckie, Ken Anderson, and Donna Dupuis. Toward a competitive pool-playing robot. *Computer*, 41(1), 2008.

[8] MT MANZURI SHALMANI. Roboshark: a gantry pool player robot. In *Proc. of the International Symposium on Robotics*, 2003.

[9] Christopher Archibald, Alon Altman, and Yoav Shoham. Analysis of a winning computational billiards player. In *IJCAI*, volume 9, pages 1377–1382, 2009.

[10] Jens-Uwe Bahr. A computer player for billiards based on artificial intelligence techniques. *no. September*, 2012.

[11] Will Leckie and Michael Greenspan. Pool physics simulation by event prediction 1: Motion transitions. *ICGA Journal*, 28(4):214–222, 2005.

[12] Will Leckie and Michael Greenspan. Pool physics simulation by event prediction 2: Collisions. *ICGA Journal*, 29(1):24–31, 2006.

[13] Julien Ploquin. *Simulateur de billard réaliste*. PhD thesis, Université de Sherbrooke., 2012.

[14] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[15] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[17] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[18] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[19] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[20] Sélim El Mekki. Carom recommender system. `https://github.com/elselim2/Recommender-system-for-the-billiard-game`, 2018.

[21] David Louapre. Théorie du chaos et effet papillon. `https://sciencetonnante.wordpress.com/2018/02/16/theorie-du-chaos-et-effet-papillon/`, 2018.

[22] Bruce Sherwood. Vpython. `https://github.com/vpython`, 2012.

[23] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[24] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[25] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *CoRR, abs/1707.08817*, 2017.

[26] Christopher Archibald, Alon Altman, Michael Greenspan, and Yoav Shoham. Computational pool: A new challenge for game theory pragmatics. *AI Magazine*, 31(4):33–41, 2010.

[27] Thomas Nierhoff, Kerstin Heunisch, and Sandra Hirche. Strategic play for a pool-playing robot. In *Proceedings of the IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2012.

[28] Michael Smith. Running the table: An ai for computer billiards. In *Proceedings of the national conference on artificial intelligence*, volume 21, page 994. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[29] Jean-François Landry and Jean-Pierre Dussault. Ai optimization of a billiard player. *Journal of Intelligent and Robotic Systems*, 50(4):399–417, 2007.

[30] Yan-Bin Jia, Matthew T Mason, and Michael A Erdmann. Trajectory of a billiard ball and recovery of its initial velocities, 2011.

[31] Inhwan Han. Dynamics in carom and three cushion billiards. *Journal of mechanical science and technology*, 19(4):976–984, 2005.

[32] Damien Ernst. Optimal decision making for complex problems, course notes, 2018.

[33] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[34] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[35] Matthias Plappert. keras-rl. `https://github.com/keras-rl/keras-rl`, 2016.