A robust controller for a two-layered approach applied to the game of billiards

Jean-François Landry, Jean-Pierre Dussault, Philippe Mahey

A robust controller for a two-layered approach applied to the game of billiards

Jean-François Landry[b,a,*], Jean-Pierre Dussault[b], Philippe Mahey[a]

[a] *ISIMA - Campus des Cézeaux, BP 10125, 63173 Aubière CEDEX, France.*
[b] *Département d'Informatique, Université de Sherbrooke, Sherbrooke (Québec), J1K2R1, Canada.*

Abstract

Planning issues in a continuous domain in the presence of noise lead to important modeling and computational difficulties. The game of billiards has offered many interesting challenges to both communities of AI and Optimization. We present a two-layered approach consisting in a high level planner and a low level controller. We propose here a refined controller for billiards based on robust optimization combined with specific adjustments to take advantage of the domain knowledge. A multi-objective formulation of a robust controller will be presented to provide the tools needed to execute any desired shot on the table, as part of a two-layered approach for the game of billiards. Some results will be then shown, followed by a short discussion on future work.

1. Introduction

A lot of research is currently done in the AI and optimization community on game-related problems, often in the form of robotics (Robocup challenge [12]) or computational games competitions (ICGA [16] and AAAI [1] conferences). Some of these games hide very complex problems which humans are often able to decompose very quickly but demand an important analysis if to be modeled and solved by a computer. It is our belief that some of the most interesting challenges can be found in the mixed continuous-discrete stochastic domain problems, and in one example in particular, billiards.

In everyday world, many of the variables affecting our decisions are continuous, like the speed at which we drive a car, or the time spent to reach a destination. Others are discrete, like the fact that we need to stop at one or two places before our final destination. We usually benefit from a lot of knowledge which was previously acquired to help us plan, and it is not an easy task to

*Corresponding author
Email addresses: Jean-Francois.Landry2@USherbrooke.ca (Jean-François Landry), Jean-Pierre.Dussault@USherbrooke.ca (Jean-Pierre Dussault), Philippe.Mahey@Isima.fr (Philippe Mahey)

design efficient planning algorithms, as it is to gather this knowledge and organize it in an intelligent manner. As such, planning in a continuous domain with noisy parameters, which is the case of billiards, usually requires a great deal of analysis and domain knowledge. Adding a notion of continuity to the possible actions, making them infinite, and considering the resulting state space to be infinite as well, greatly complicates things. If we picture a planning in such a domain in which one wishes to plan a sequence of actions to reach a specific goal, our search will quickly become intractable.

If approaching these types problems from a human's point of view, a simple way to see things would be to divide them in two layers: planner and controller. Indeed, when trying to devise a plan for a series of tasks to perform, it is not necessary to plan every little detail with our planner. A simple general assessment of the best targets to aim for to minimize the chances of mistakes might be enough to provide us a general plan to proceed. We can then consult our controller that will compute all the necessary details (strike force, angle...) to provide us with precise information at base-level and let us know of the real value of such a plan, and if indeed viable. Of course, not every domain can be easily approached, and simply building an accurate and fast-enough simulator is another problem of its own.

We will focus in this work on one particular well-known game and hobby, billiards. A lot of work has already been published on this topic, usually on the of the aspects of robotics ([22], [14], [20], [11], [25]), physics ([19], [6], [30], [29], [21], [24]) or player AI ([27], [10], [3], [18]). The latter, notably an automated decision-making process for this game, is the one which we will further discuss.

Billiards is a very interesting and challenging game, characterized by the presence of a theoretically infinite set of possible shots, each of which leading to a different outcome, thus creating a vast continuous search space. One must also account for the opponent's shots, by either creating an environment in which he will be able to keep the turn, or leave the opponent to play a very risky shot.

The physical aspects of the game must be simulated to a high level of accuracy if hoping to one day apply the solutions found to possibly challenge real-world professional players. Creating such a simulation also comes with its share of problems. Although many game simulators currently exist on the market, it is unclear as to how faithful these are to real world physics. Aspects like ball-ball collisions or ball-rail collisions are often simplified to create something visually pleasing, but not always physically accurate.

For the moment, we take for granted the availability of a physical simulator, which will be described briefly, while still keeping in mind that the approach used should be customizable to suit specific changes in the simulator.

It is our goal to explore this category of problems, and propose a solution using optimization methods, to search the state space in an intelligent manner by using knowledge deduced from the problem. The novel contribution we propose consists in the following aspects: a redesign of a previous optimization model to account for multi-objective shots to complete the array of tools needed by the controller, a robust formulation of this model to account for the stochastic aspect of the game, and finally a proposition of a general two-layered planning

approach where we single-out the controller component with evidence of its good performance.

In the first part of this paper, we will do a recall on the general problematic surrounding the game of billiards to single out the research aspects on which we wish to expand. We will present some refinements to a previously defined optimization model ([18]) for the game of billiards. We will next take a closer look at the specifics of the game, to determine a list of tools needed by the controller. We will then proceed to the modeling of a robust form of the problem we are trying to solve to account for the stochastic nature of the game, to finally conclude with some results and future work.

2. The game of billiards

Of the many existing billiards variants, we choose here to take a closer look at the game of 8-ball. This specific variant has been the focus of previous work in [18] [5] [14] [19] [26]. It has also been used as a testing platform for various agents in three past computational tournaments. Although the challenges present in the various types of billiards games vary, the aspects we discuss here do not.

The game of 8-ball is one of the most commonly played variants in North-America. It consists of 15 numbered balls, divided in a set of high (1-7) and low (9-15) balls as well as the 8th ball. To win at 8-ball, a player must successfully pocket all of his balls, high or low, and pocket the 8^{th} ball last. If the 8^{th} ball is pocketed before by accident, the player loses the game. Each shot must be specified (ball, pocket), and failure to complete the shot as called will result in a loss of turn. The first shot (break) is decided in a random fashion, and if a player pockets a ball on the break, he may then choose for which balls he will play. If no balls are pocketed on the break, the turn and choice is given to the opponent. A detailed description of the specific rules can be found on the site of the Billiard Congress of America ([7]).

It is possible to view this game as a combination of two fundamental aspects. The first and possibly most important of these aspects is related to the technical parameters needed to sink a ball in a pocket. Indeed, once a ball and pocket have been selected, a player must find out what are the optimal parameters to successfully complete this shot. If a player doesn't have the ability to execute such a shot, he will fail to advance in the game. The second aspect of the game is one that applies to better players, and that also makes the game such an interesting challenge; the ability to plan a sequence of shots. When a player executes his first shot, if he's fairly confident of his success, he will gain a great advantage carefully repositioning for his second shot. He must however analyse and decide which position to reach, and which order would represent the optimal sequence of balls to pocket. Such an analysis will depend on various parameters, namely the technical level of the player, his weaknesses and strengths in executing particular shots. In various games, other parameters like points and faults will come into play, of which we will make abstraction in this paper since these will need to be addressed at the planning stage of the process.

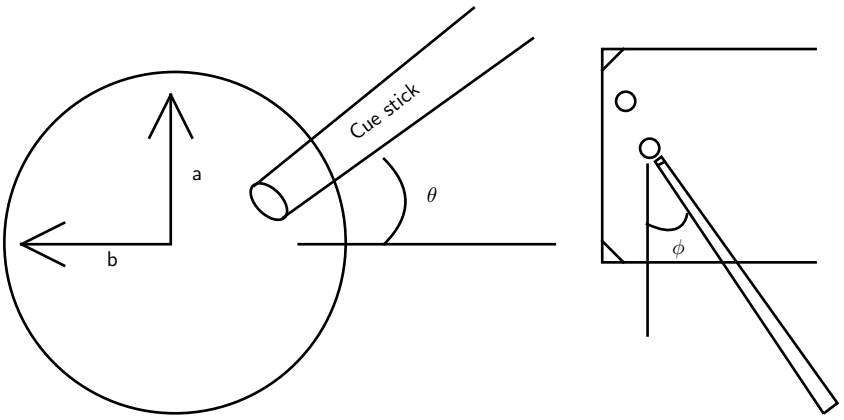

Figure 1: Side view on left figure to illustrate shot parameters. Top view on the right figure to illustrate cue angle parameter (ϕ).

2.1. Physics simulation

The poolfiz simulator ([19]) used in previous competitions allows for a two dimensional physics simulation of a pool table, by simulating the various ball interactions on the table following shot execution. The analytical solution of a quartic equation describing the ball trajectories on a table allows for a direct and precise computation of impact times, as well as a very fast simulation since no approximation methods are used.

The format used to specify shots on the simulator corresponds to the impact force transmitted to the cue ball by the cue stick, and is translated to the following 5 parameters(Figure 1):

- a and b represent the horizontal and vertical offset from the center of the cue ball;
- θ the elevation of the cue stick relative to the horizontal plan of the table;
- ϕ the orientation of the cue stick (in degrees) relative to the table;
- v, the initial velocity given to the cue ball.

By feeding these parameters to the simulator, the resulting table state is given, and an analysis of the shot's success or failure, as well as its value, may be computed. Since the simulator is deterministic, the same results will always be returned given the same input parameters are used.

2.2. Modeling the game of billiards

A defining and very important aspect of billiards is the principle of turn-taking (see [5]) in the game. It corresponds to the fact that in billiards, one can play all the way until the end of the game if he doesn't miss a shot, but he's

also got the possibility of giving the turn to his opponent by the execution of a safety shot, if he thinks this global strategy will be beneficial in his overall game-plan.

In reality, defensive play is one of the most interesting and difficult aspects of the game of billiards. It demands a very precise analysis of the game state to weight the value of doing an offensive shot versus a defensive one. If a shot is too risky, it may be more valuable to play defensively, but if no good defensive shot is playable, one might as well take his chances and continue offensively. This kind of situation is easier to see in the games of Snooker and Straight, where it is often possible to have long exchanges of defensive shots, until one opponent sees an opportunity for a great shot, or one of the players misses his defensive shot.

The general model proposed in [5] defines the winning strategy as a perfect Markov equilibrium between two players. This implies the existence of a value function v on the state space such that

$$v(s) = \max_u [\int_S v(s')p(s'|s,u)]$$

where $s \in S$ is the state of the table and p is the probability to reach state s' when applying action u in the state s. Observe that, on his side, player 2 will minimize the same value function, but the model only considers offensive strategies defining sequences of successful shots until the player loses his turn. A defensive strategy should be an additional decision to leave the table in the worst possible state for the adversary and this could be done if the value $v(s)$ is below some given threshold (high risk of missing the next shot). We will see in section 2.5 that the decision variable can be resumed by the repositioning target of the cue ball. To implement a winning strategy following this model is obviously very costly and simplifications have to be made to drive the search for the 'best' shot on a reduced state space to minimize the simulator calls.

2.3. Model specification

If we take a closer look at the model in [5], the value of a shot in a given state is defined as:

$$v'(s) = \begin{cases} \max_a \left[\int_S v(\cdot)dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 1 \\ \min_a \left[\int_S v(\cdot)dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 2 \end{cases}$$

We start in state s^0 and the active player defined by $\lambda(s^0)$ specifies an action u^0. The next step s^1 is determined by the transition function associated with probability $p(s^1|s^0,u^0)$ and the game continues with $\lambda(s^1)$ specifying the next shot. The game continues in such a way until a terminal state s^T is reached, and the winning player receives the reward $r(s^T)$ from the other player.

If we are able to find the sequence of shots leading to $\text{argmax}_a \left[\int_S v^*(\cdot)dp(\cdot|s,a)\right]$, we should finally have the optimal solution with the best chances of success.

As mentioned earlier, this model is of course very general, and although it promises an optimal solution, it is not computationally tractable unless using some form of simplification.

A classical and popular approach to this problem is to discretize the search space, and use a Monte-Carlo sampling approach to explore the domain in a timely manner. This approach was first explored with great results by Smith M. in [26] and then refined in [3]. Although seemingly effective for the game of 8-ball, it is unclear as to how well such an approach would do when faced with more complex variations of billiards games. Indeed, one of the assumptions made for that approach was that a safety shot in 8-ball would never be made unless all other offensive shots were discarded. However like mentioned earlier, in games like Straight or Snooker, safety shot exchanges are more than just common, they are almost always present in every game. Proceeding with a discretization approach in such a situation might prove problematic since the breadth of the search space will quickly explode. It is our impression that instead of exploring the state space, a lot of time could be gained by rather exploiting the game knowledge to our advantage, and thus eliminating useless calls to the simulator.

Evidently, exploiting the knowledge of the game also has its downsides. One must be very careful not to fall into the trap of encoding each and every possible situation as a rule-based expert system, since it may become very hard to decide with confidence if a shot really is better than another one. However some basic pool knowledge can easily be applied when analysing a table to quickly discard useless shots, as discussed in [18].

3. A two-layered approach

If we have to look at any professional tournament game, we can observe that a player will never execute a shot to which he doesn't know the exact outcome. Unless trying to break a cluster, each and every shot will be carefully planned to maximise the chance of a good reposition, and even when going for a cluster, it will usually be a controlled shot aimed to separate one or two balls from the pack and continue playing. It is very seldom that other balls than the cue and target balls will move on the table since the more collisions we have the harder it becomes to accurately predict the resulting trajectories, especially when noise is present in the shot. There is no reason why it should be different for a computer simulation. Thus if we are able to carefully determine the best positions to reach on the table for the next shots, we should be able to easily find a shot sequence maximizing our chances of winning.

For our model, this adds up to the following points:

- Discretization of the domain using repositioning targets.
- Partial ordering of the targets using shot difficulty coefficient.
- Listing of best shot sequences by planner.
- Consultation of the controller to discard bad shots and compute success percentages.

- Reorganization of shot sequences in order of success percentages.

This decision process is illustrated in the simplified Figure 2. It is important here to note that the shots which are generated by the planner correspond to a ball-pocket-target combination, rather than specific shot parameters. The shot parameters themselves will be found by the optimization procedure.

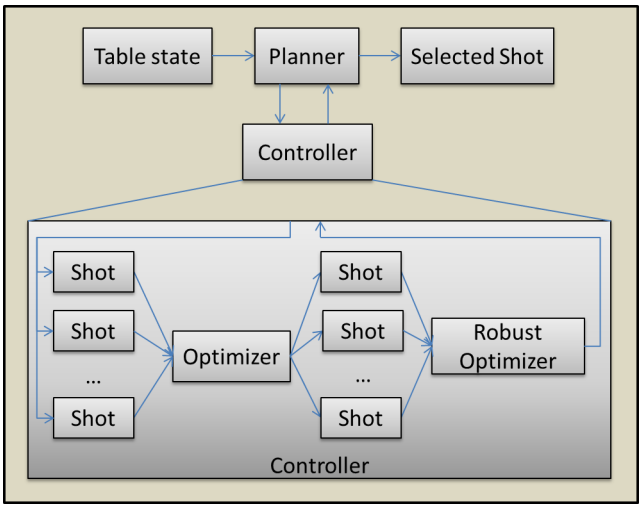

Figure 2: Illustration of a two-layered approach, where the planner first elaborates a plan by considering the best targets on the table and generating various shots to optimize, to then consult the controller, which in return will provide the list of achievable solutions.

We hope to differ from the planning approaches used in [26] and [3] by using a technique that first generates a plan based on interesting positions on the table, to then explore the best of these plans in a robust manner. The planner will then adjust the initial plan with the updated information and decide of the optimal step to be taken. The contribution we make in this paper mainly relies on the improvement of the controller aspect of the game, since the planning aspect will have to be adapted on the type of game being played, as well as on the stochastic aspect introduced by noise addition to the shot execution. We wish however to stress the importance of the robust controller in relation to the planner in the presence of noise-induced parameters. In a deterministic game, the use of a planner might be rendered useless (discussed in section 5) since a strong controller will most likely always find a shot, but with a stochastic game, the solution found will not always be robust and thus not always optimal. We still provide a few leads in the last section as to the direction we will take in future work to plan in an efficient way.

To illustrate the necessity of a planner in some situations, and to motivate the existence of a two-layered approach, we provide in figures 3 and 4 an example of two possible shots leading to two completely different positions on the table. For these two shots, gaussian noise was added to the shot parameters using the

standard deviations of the past Computational Pool Olympiads (ϕ=0.125 deg, θ=0.1 deg, v=0.075 m/s, a=0.5mm, b= 0.5mm). Although in most cases it would seem that a very good controller on its own can compensate for the lack of planning in the game of 8-ball, in this specific case, if the shot with the best position is chosen (in relation to pocket distance and angle), it results in a much harder next shot. Both shots are robust to noise (succeed in pocketing the first and second 100% of the time) but over an average of 500 games starting from this state, the shot 1 in Figure 3 led to victory in only 56% of the cases, while if the second best shot was selected (shot 2 in Figure 4), its different position on the table led to a 96% success rate. The reason for this is simple, if shot 2 is selected, it is easier to then follow with shot 1 and stay in good position for the 8th ball since it is in the same half of the table. If shot 1 is selected first however, a long shot has to be made to reach the next ball, thus failing more often. This is an isolated case not often encountered in 8-ball but one that serves as a reminder that a strong controller cannot always account for the lack of planning.

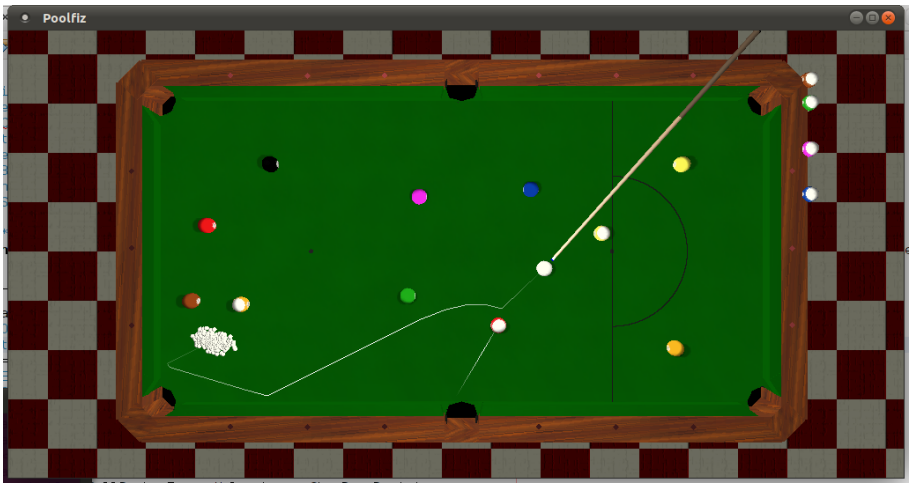

Figure 3: Shot 1. Robust shot leading to best possible position on the table in average. Success percentage to terminate the game: 54% (when testing with 500 states resulting from noise-induced parameters). White dots represent the various cue-ball position reached for noise-induced shot.

The following sections address the controller aspect of the game which will execute the shots requested by the planner and return the resulting robust shot with a success percentage.

4. Controller

The fundamental aspect of our approach consists in actually developing each and every tool a good player would have at his disposal. By this we intend to create a player with the ability to find out exactly what parameters to use for

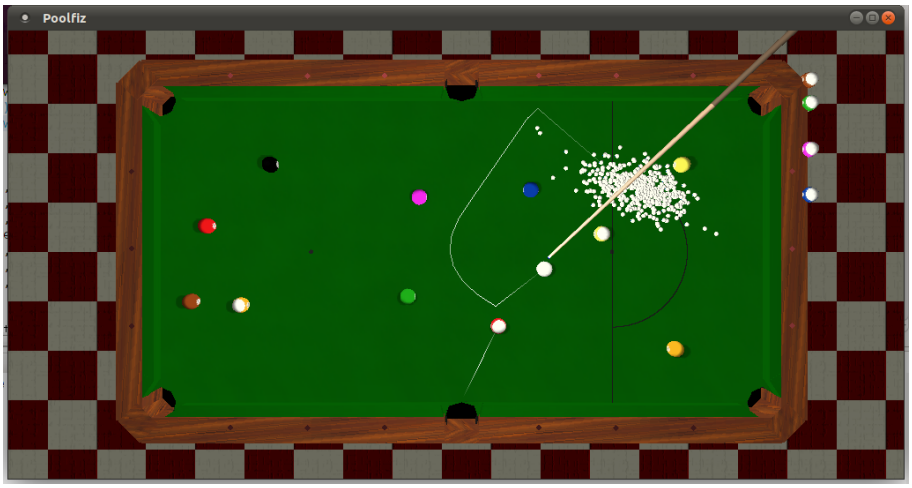

Figure 4: Shot 2. Robust shot with a slightly lower value than in shot 1. Success percentage to terminate the game: 96% (when testing with 500 states resulting from noise-induced parameters). White dots represent the various cue-ball position reached for noise-induced shot.

a shot, if such a shot is possible. Thus a part of this work consists in a pre-analysis of a table state which will inform us on the targets to reach, and of a formulation of the problem as a least-square problem to solve by minimisation techniques.

4.1. Multi-objective shots

In this section we will revise a previous optimization model to account for the problems of breaking clusters, and doing multi-objective shots, which can be quite useful in some variants of billiards games.

4.1.1. Direct shots

We first explored the aspect of position play in [13] but reiterate here the basic principles, from which we will derive other useful techniques for our player. To compute a direct shot, we start by performing a quick analysis which allows to determine at which point the cue ball should hit the target ball for it to reach the pocket. This is only an estimate, based on the 90-degree rule (see [2]), but allows us to find a suitable starting point for our optimization procedure. We can also eliminate shots for which the angle isn't achievable.

We can then proceed to define an objective function, which corresponds to the position of the cue-ball at its final resting time following a shot. We add a constraint to this function, representing the fact that we want to sink the target ball in a given pocket. However since this constraint will likely not be solved when we start optimizing our function, we model it as a penalty added to our objective function. This allows us to find a suitable solution to pocket the target ball first, and then proceed to find the best possible parameters to

get us as close as possible to the target position. Assuming vector x contains the values related a given shot on the simulator, we can formulate the following objective function to minimize:

$$\min_x \frac{1}{2} \left(\|\mathbf{p}_c(x) - \mathbf{c}\|^2 + \rho \|\mathbf{p}_o(x) - \mathbf{p}\|^2 \right)$$

where $\mathbf{p}_c$, $\mathbf{p}_o$ represent the outcome of a call to the simulator, used as a black-box for the moment, with the vector of shot parameters x (corresponding to $[a,b,\theta,\phi,v]$ in section 2.1). More precisely, $\mathbf{p}_c$, $\mathbf{p}_o$ are the positions of the cue and target ball at final resting time, $\mathbf{c}$ the target position we wish to reach with the cue ball, $\mathbf{p}$ the pocket at which we aim and ρ a penalty value to insure our target ball is pocketed (fig 5).

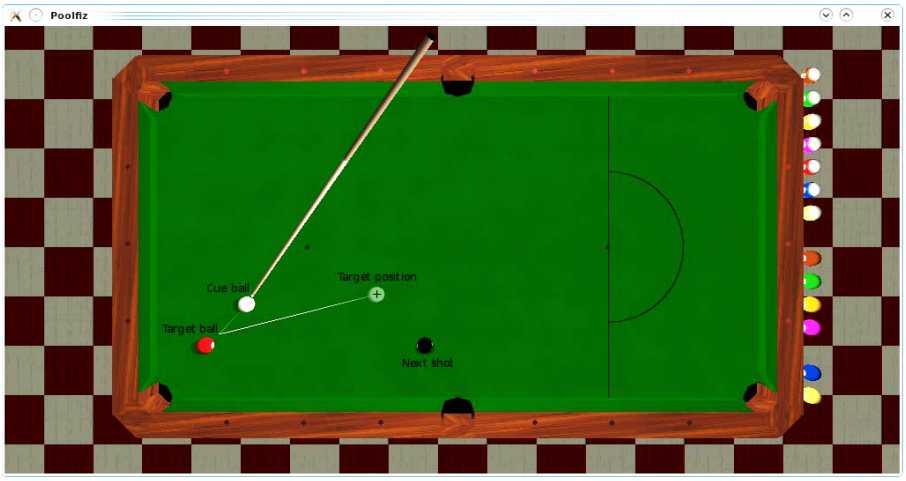

Figure 5: Shot optimisation for repositioning on a specific target for the next shot.

4.1.2. Breaking clusters

We have already discussed the aspect of breaking clusters in ([18]) using pre-computed tables to facilitate function optimization. The approach used made it possible to break clusters but not in every situations, and the success of the optimization was highly dependent on the starting points. As such, we proceed to model our objective function in a different and more efficient way. We first re-use our initial model in (4.1.1), but instead of aiming for a repositioning target on the table, we switch this target to aim for a cluster target. We divide our objective function in two parts. The first part will be to satisfy the constraint of pocketing the target ball and to try to reposition at the center of the cluster we are trying to move. Obviously, trying to replace the cue ball into a spot where another ball is laying is quite difficult, and impossible in most cases. However, we can attempt to aim for that position, and as soon as we get to a collision with our target, switch to the second part of our objective function,

which includes a minimum velocity for this collision, thus dispersing balls and hopefully obtaining a better table state.

We get:

$$\min_x \frac{1}{2} \begin{cases} \left(\|\mathbf{p}_c(x) - \mathbf{c}\|^2 + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2\right) & \textit{if i=1} \\ \left(\text{obj2}(x) + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2\right) & \textit{otherwise} \end{cases}$$

with $\text{obj2}(x) = \|\mathbf{v}_c(x) - \mathbf{v}_{\min}\|^2$ as our second objective, where $\mathbf{v}_{\min}$ represents a minimum velocity, and $\mathbf{v}_c(x)$ the velocity at time of impact. i represents a boolean value, which is set to true if there is an impact with the second target. The minimization of our second objective $\text{obj2}(x)$ will ensure a minimum speed is given to the cue ball at impact time, dispersing our cluster. We do not specify here a $p_c(x)$ since we don't mind if the point of impact changes, as long as the impact occurs at a specified minimum speed.

Another problem, which arises when optimizing a function with many clustered balls in the table state is the time needed to do the simulations. For example, using poolfiz, a break shot demands 300 times more time to simulate versus a single impact shot. Our optimization methods also suffer when too many events occur after a shot since the function becomes very chaotic and the slightest variation in the parameters will result in a totally different table state. As such, it is not worth trying to optimize such a function considering the computing time constraints, and to work around this issue, we can proceed to a very simple trick. If two or more balls are together in a cluster, we can aim for the center of this cluster, and remove all other balls on the table. Once a solution is found, we can put the balls back on the table and test our shot. Of course this approach doesn't guarantee that the solution found will always be usable, but it is our experience that in most cases it easily breaks the cluster and demands little computing time.

4.1.3. Double shots

Having just defined a way to execute complex shots to break clusters on the table, we can use this secondary objective, and modify it to instead pocket a second target ball if desired. Indeed, by simply redefining $\text{obj2}(x)$ to $\text{obj2}(x) = \|\mathbf{p}_{o2}(x) - \mathbf{c_2}\|^2$ in our previous model, we will try to send our second ball (defined by position $\mathbf{p}_{o2}(x)$) to the desired pocket $\mathbf{c_2}$, as can be seen in Figure 4.1.3.

It is possible to see that we removed our cue ball repositioning constraint, since it most likely will be very hard or even impossible to control the cue ball final position while having constraints on two previous hits. Of course, only a limited number of impacts may occur before our ball loses all of its velocity, but results show that it is easily possible to find the parameters of a two-ball shot if one exists. This particular kind of shot is of little interest in the game of 8-ball, since it usually proves too risky and unnecessary. However, in the game of 9-ball for example, one player is always constrained on hitting a specific ball first, and this type of shot is exactly the kind that might allow the player to win the game early.

It may also be important to note that the current version of the poolfiz simulator doesn't impart the correct velocity after ball-rail impacts. This greatly

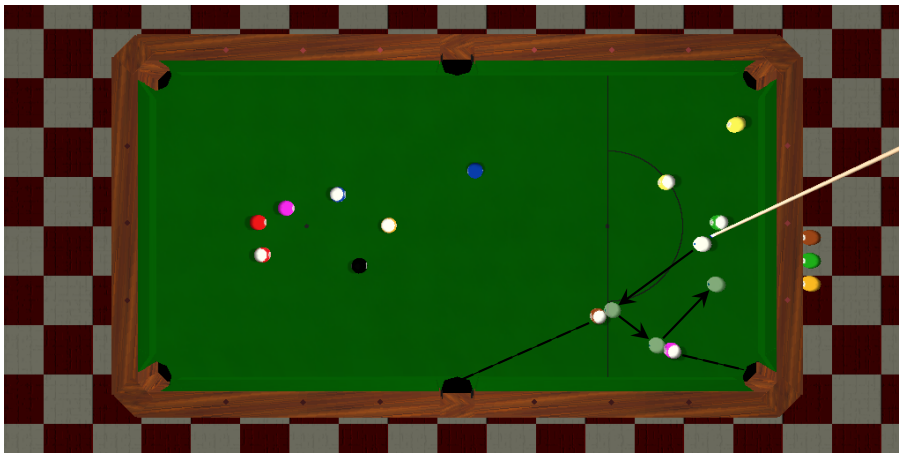

Figure 6: Double shot execution, where the first target ball is pocketed to the center pocket and the second target ball, to the corner pocket.

reduces the ability to reach various targets on the table, and also prevent fancier shots otherwise possible on a real table. It will be interesting to test our optimization model on a more faithful simulator to see the improvement in constraint satisfaction, and hopefully new interesting shots.

4.2. A robust model

The initial model developed was targeted at providing our AI player with the tools necessary to reposition anywhere he liked on the table. This feature remains quite powerful and very desirable in that it is a very efficient way of finding shots with good repositioning value. The strength of this approach is also illustrated by the fact that a player is able to finish virtually any random table state he is given.

This model, however, tends to lose most of its advantage in presence of noise-induced parameters. As is discussed further in section 5, the success rate fell far down when noise was added to the shot parameters. Two main hypothesis can actually be made as to the explanation of these results; either the wrong shot is selected amongst those generated, such that the shot chosen will have a completely different outcome with the slightest amount of noise, either the shots generated are riskier to achieve better repositioning, resulting in a higher sensitivity to noise.

If the wrong shot is selected, it is simply a matter of adjusting the evaluation function that is used to compare the available shots. This can be solved by launching many more mini-tournaments to determine statistically the best weights to be given to each computed shot. This calibration will need to be done one way or another.

If instead the problem lies within the shot generation model, an adjustment will be needed to our objective function. In the current model, the goal of aiming

for pre-defined target for the next shot works very well without noise. However it is possible that the strength of the player (i.e. his ability to reposition almost anywhere on the table) is also his greatest weakness. In hope of reaching a given target, it is possible that the shots selected will be very fancy, that is, will have strong spin effect. Actually in the current model, it is not possible to measure this sensitivity unless we do sampling after a solution has been found to the optimization model. This is not very helpful except in the case of shot selection, since it will only help to evaluate the shots which were computed, but will not provide us with more robust ones. This problem is illustrated in figures 7 and 8, where the white dots illustrate the type of shots we want to favour. We can roughly estimate a shot difficulty by measuring the distance and angle between the balls and pocket and this is useful in selecting good positions on the table, but shot difficulty also depends on the parameters used to execute the shot. As such, we need a way to find robust shot parameters, to sink the object ball, and also retain a decent position for our next shot.

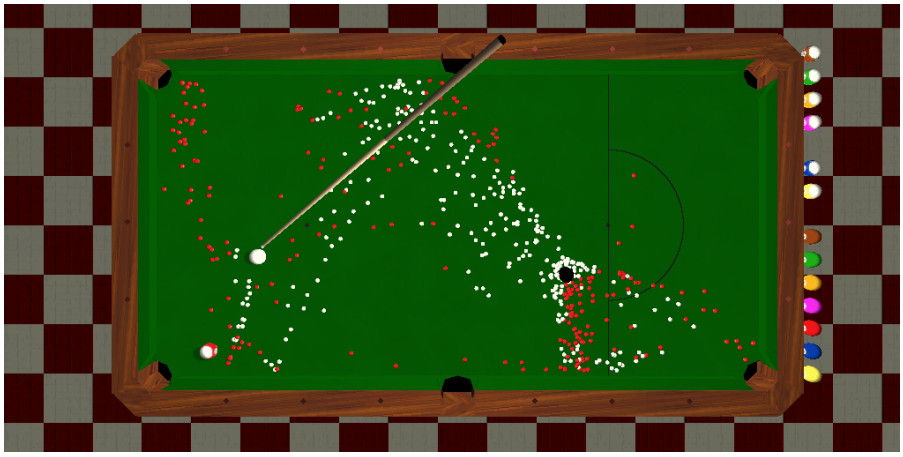

Figure 7: Illustration of possible repositioning targets, and sensitivity to noise when trying to pocket the ball in the bottom corner pocket using various shot parameters. White dots represent positions which are attainable while still pocketing the target ball when noise-induced parameters are used (90 to 100% of the time). Red dots represent positions which are reachable, but don't guarantee we will successfully pocket the target ball (success rate is between 70 and 90%).

4.2.1. Robust counterpart

Robust optimization, as detailed in [9] and [8], can be defined as a methodology to treat uncertain problems usually formulated as:

$$\min_{u}\{f_0(u,\zeta) : f_i(u,\zeta) \in K_i, i = 1, ..., I\} \tag{1}$$

where u is the vector of decision variables, f_0 the objective function, and $f_1, ..., f_i$, the structural elements of the problem (or constraints), and ζ the

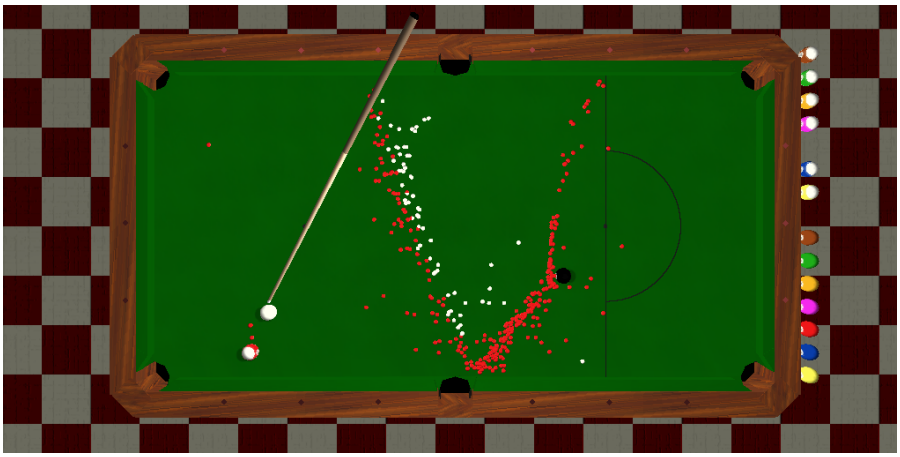

Figure 8: Harder shot where we can see the sure shots have considerably diminished. The distribution of the reachable zone is most likely affected by bottom center pocket which would result in a missed shot when noise modifies the cue ball trajectory and directs it in the pocket.

data of a particular problem instance. In our problem, this would correspond to the objective function of maximizing the value of the position, f_0, under the constraints of pocketing the balls $f_1, ..., f_i$ as described in model (6), with the vector ζ representing the shot results with noise-induced parameters.

Many methods can be used to solve robust optimization problems, unfortunately most of them are not directly applicable to our case since for the moment we use the simulator as a black-box and can't take advantage of any function-specific knowledge. However, we can still use some basic principles to find a good solution if one exists.

The first aspect which needs to be addressed is the value of a specific repositioning target. When choosing shot parameters, it is possible it will become necessary to trade-off the repositioning value to a certain extend. It is, however, also possible it will not be affected at all. This is why it would be better to aim for a shot remaining in a good zone rather than at a specified position. A good model should include two fundamental scalable variables; a shot success probability α_1, and a repositioning success probability α_0.

Again, taking our initial model, using balls' final resting positions as $b_i(u)$, with the vector u representing our initial shot parameters and $b_{\text{cue}}(u)$ to represent our cue ball. Let's use p as our pocket position for pockets 1 to 6.

The noise-less model, or as we will call it here a nominal model, can be defined as a function minimizing the distance between the cue-ball's final position and a given target t on the table, under the constraint of pocketing the target ball.

$$\begin{aligned} &\min f(u) = ||b_{\text{cue}}(u) - t|| \\ &\text{s. t.} \\ &\quad ||b_i(u) - p|| = 0 \end{aligned} \tag{2}$$

As we just mentioned, this model doesn't hold very well with noise-induced parameters. To counter this, it is possible to redefine it instead as:

$$\begin{aligned} &\min f(u) = \text{Posval}(u) \\ &\text{s. t.} \\ &\quad ||b_i(u) - p|| = 0 \end{aligned} \tag{3}$$

The objective here will be to minimize the value of a reposition Posval(u) (with a smaller value representing a better position), while still satisfying the same constraint of pocketing a ball. The fundamental difference with this parameter is that we no longer try to reach a precise position on the table, but rather maximize the value of a position in a zone for the next shot. The problem with this approach is the discontinuity of the function. Indeed, many positions will have a constant value of 0, and will create stationary points from which it will be hard to climb out. However it is possible to use a two-step approach, by first minimizing the distance to a target using model (2), to then switch to (3) when we have reached the desired zone. The goal of this reformulation of the initial problem is to introduce a model for which we will be able to formulate a robust counterpart. Indeed, if we wished to robustify model (2), we would stray from our initial objective which is to reposition in a decent zone for our next shot. By using model (3), we are able to relax our objective function so that it becomes easier to minimize under a robust constraint.

Assuming our perturbed shot parameters(for a given noise) vector ζ. If we consider U as the set of the normal range of the parameters under which our shot still remains successful $U = \{\zeta : b_i(u, \zeta) = p\}$,

We can formulate the comprehensive robust counterpart of (3) as:

$$\begin{aligned} &\min_u \sigma \\ &\text{s. t. } \text{Posval}(u, \zeta) \leq \sigma + \alpha_0 \text{dist}(\zeta, U) \\ &\quad ||b_n(u, \zeta) - p|| \leq \alpha_1 \text{dist}(\zeta, U) \end{aligned} \tag{4}$$

$\forall \zeta$. Thus, our minimized value σ will represent a robust solution, with constraints and objective weighted by the adjustable α values, and the noise effect on the satisfaction of our constraint. Of course, to solve this model, we will have to proceed to a discretization of our normal range values U, but it should provide us with a good estimate of a shot's chances of success. It should be noted here that we used a comprehensive robust counterpart to describe our problem, since it is possible to imagine a solution where the constraints (pocketing the ball) will not be satisfied for every ζ in U. This would result in a possible imbalance where a shot with guaranteed success but a guaranteed useless position, would be chosen over a shot with 95% success rate and very good reposition.

4.2.2. Penalized robust model

As was needed for the earlier described position-play model in section 4.1.1, we will now proceed to a slight reformulation of the robust model (4) to consider the constraints as penalties to our objective function. Solving our problem in

this manner allows us to optimize a function that will not only maximise the value of the position reached, but also minimize the distance between the target ball and the pocket so that when searching for an optimal solution, we remain in the region closest to satisfying the constraint of pocketing this ball.

$$\min_u f(u) = \max(\text{Posval}(u,\zeta) - \text{Val}_{\min}, 0) + \rho||b_n(u,\zeta) - p||_\infty \ \forall\zeta \qquad (5)$$

By finding the parameters of u which minimize this function, we should be able to reach a solution that is not only robust, but also leaves us a very good zone for the next shot. We use the infinity norm on our constraint as to penalize it in a linear fashion to facilitate its optimization, and try to reach a position with a minimum value $\text{Val}_{\min}$ for the zone.

5. Results

We present in this section the result of various tests, mainly to illustrate the potential of the proposed approach, but also to show the progress which has been made in regards to alternative methods used to solve the game of billiards. As has been mentioned before, it is very hard and complex to design a physically-accurate simulator for the game of billiards. We are currently working on such a simulator [1], in hope of having it carefully and precisely calibrated to a real billiards table. In the meantime, however, two options are available to us; the Poolfiz ([19]) and the Fastfiz ([15]) simulators. Poolfiz was used as the basis for the first three computational pool tournament. Subsequently, Fastfiz, a re-engineered version of poolfiz, was developed by the Stanford Computational Billiards Group ([15]) to be used in future competitions. Fastfiz has the advantage of being must faster than poolfiz, and as such, allows for more computations in the same time, and for such reason we made the necessary conversions in PoolMaster to use it. A deep and well-described analysis of the winning player of the last computational pool olympiads ([3]) allows us to get an idea of the level at which it currently stands. Since we are in preparation for the next tournament, we will do some comparisons here with the another player (Pickpocket) that will not be participating this year and which was made available to us by its author [2], and was also used for comparisons in [3]. It is important to keep in mind that these comparisons are simply to show the validity of the robust controller model proposed here. The true benefit of such a model may be more apparent in other forms of billiards (9-ball, snooker, straight), but as we show here, even the robust controller on its own performs relatively well for the game of 8-ball.

For these tests, Pickpocket is used exactly as provided by the author, using the default settings, where a search breadth of 15 shots is done at the first search

[1] Julien Ploquin, forthcoming Master's thesis, Université de Sherbrooke

[2] We wish to thank Mike Smith for graciously providing us with his player, Pickpocket, to perform various tests.

level. The only thing which was adjusted was the noise level used for the shot evaluations, which was adjusted to the correct values depending on the noise used for the various tests.

PoolMaster was used with the optimization library NLOPT([17]), using the method BOBYQA ([23]) to minimize the objective functions. Bound-constraints were set on the shot parameters to limit shot velocity and stay within the bounds of a physically possible shot. A sample of 15 shots was done to evaluate the value of $f(u)$ in model (4).

5.1. Random tables

To determine the efficiency of the robust approach, we generated 500 tables with balls randomly positioned, and constrained the player to play on stripes on each of these table states. In such a way, we hop to minimize the variations which may occur from a table state to another after a break shot, to better compare the two players. A maximum time of 10 minutes was allowed players to compute their shots on the same computer, to conform with the previous rules used for computational pool tournaments. Players were all given the same starting table states and played until they missed a shot, if they finished all their shots, this counted as a success, otherwise as a failure. We justify the sample size of 500 table states by pointing at the comparison graphs. These show a cumulated average of the successes for cleaning the table at different noise levels. It can be seen that the average usually stabilised at a sample of around 300. We were also interested to see how the two players would fare at different noise levels. Such a study has already been done in depth in [4] to determine optimal noise level for a competition. We are not interested in reproducing these tests here, but we nonetheless performed each test at levels of 0, 0.5 and 1 of the parameters used in previous tournaments (standard deviations of ϕ=0.125 deg, θ=0.1 deg, v=0.075 m/s, a=0.5mm, b= 0.5mm) to see if the improvement of the player would be similar independently of noise levels.

To first gauge the effectiveness at which the nominal model could clean off the random table states with the added capacity to break clusters, we ran some preliminary tests with no noise added to the shot parameters. The result was a success rate of 100% for 500 games (not shown in graphs for obvious reasons), which means the player is now well-rounded enough to deal with almost any given situations on the table. By watching some of the games of these tests, we were able to see that some extraordinary, and very unlikely shots were performed in many games. These kinds of shots are rarely observed on real life games and for a good reason, they are highly susceptible to the sightless change in the shot parameters. The same success rate was observed for the robust version of PoolMaster. Pickpocket reached an average of around 90% for these tests, as can be seen in Figure 9. This means that even without noise, it is not guaranteed that any approach will always find a solution in any given state. Of course if the cue ball is isolated in a corner by adversary balls, it is possible that no offensive shots are possible. Thus it is more than likely that the robust and non-robust versions of PoolMaster performed some interesting gymnastics to find a solution in every table state.

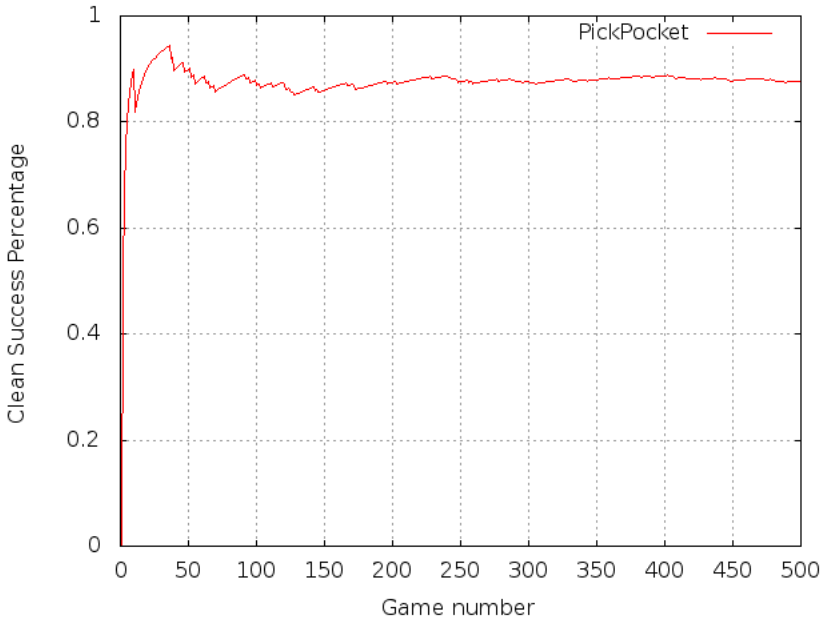

Figure 9: Average rate to clean off the table without missing a shot when no noise is applied on the shot parameters. Score for PoolMaster is not displayed since it is at 100% for the whole set.

We next proceeded to tests the players with various levels of noise. First at a level of 0.5x (in Figure 10), we saw the success percentage fall all the way down to 32% with the non-robust approach. This gives us a great deal of information concerning the problem at hand. It means that shot planning will probably not be as important as shot evaluation no matter which method is used. This agrees with the findings in [3] which states that a better solution is not necessarily found if a deeper search is done rather than a more thorough search at the first level using the same time constraints. The robust approach, however, yielded a much better average of 82% in this case, while Pickpocket reached an average of close to 66%. Although it is too early to conclude on the efficiency of the robust method, it does show some worthy potential in low noise situations.

The final tests for the random table states were performed at a noise level of 1x. These results are shown in Figure 11. The non-robust approach, predictably, performed quite poorly with an average of 14%. This illustrates the fragility of the shots generated when noise is not part of the evaluation. The robust approach of PoolMaster stabilized at a rate of 64% to clean off the game, while Pickpocket remained at around 50%. A non-negligible amelioration of 50% was achieved by using the robust model for PoolMaster. This shows great potential as any improvement to the function for position evaluation will likely provide better positioning, which at the moment doesn't take into account rail or ball

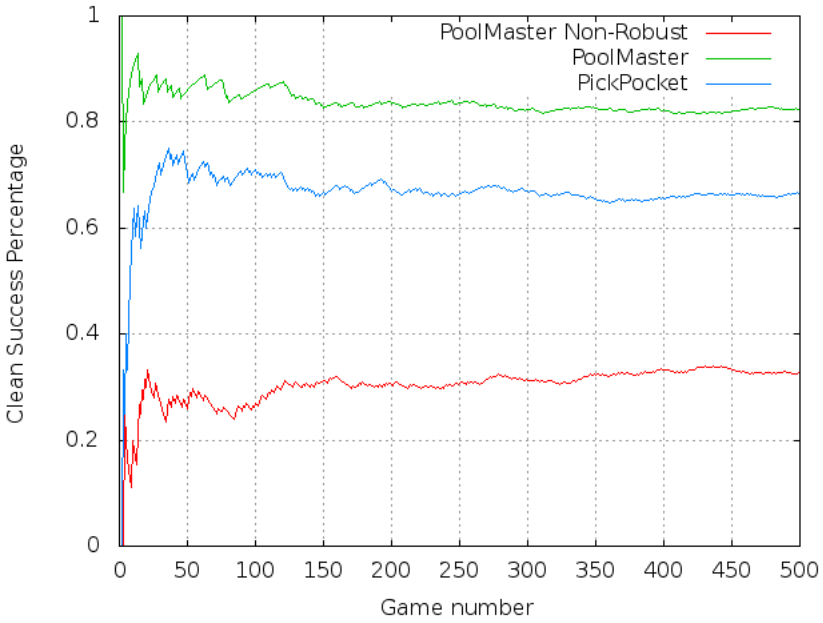

Figure 10: Average rate to clean off the table without missing a shot when a noise factor of 0.5 is used.

proximity, thus limiting the range of possible shots importantly. The difference in success rate with Pickpocket also seems to be consistent with the 0.5x noise level, thus confirming the value of the robust approach.

If we compare these results to those obtained by the defending champion Cuecard [4], the average success rate is not as high. Cuecard was able to clean-off-the-break with success close to 70% of the time at a noise level of 1x. It is important to remember however that the set-up for tests used here is not exactly the same. In [4], each player used the same break, which was we assume was the same as in [3] (the result of extensive off-line search). In this case we completely removed the break from the equation by only using randomly generated tables. It is important to remember that since players are always constrained on aiming for stripe balls, it may be possible that in given situations, the only possible shot is highly risky, thus making a perfect score very hard to reach unless using noiseless shots.

5.2. Full games

We provide here some full games between the robust version of PoolMaster and Pickpocket, in order to give a general idea of the state of the player. Since many factors come into play in a tournament like the break shots, safety shots and positions selected for ball-in-hand, these have to be taken for what they

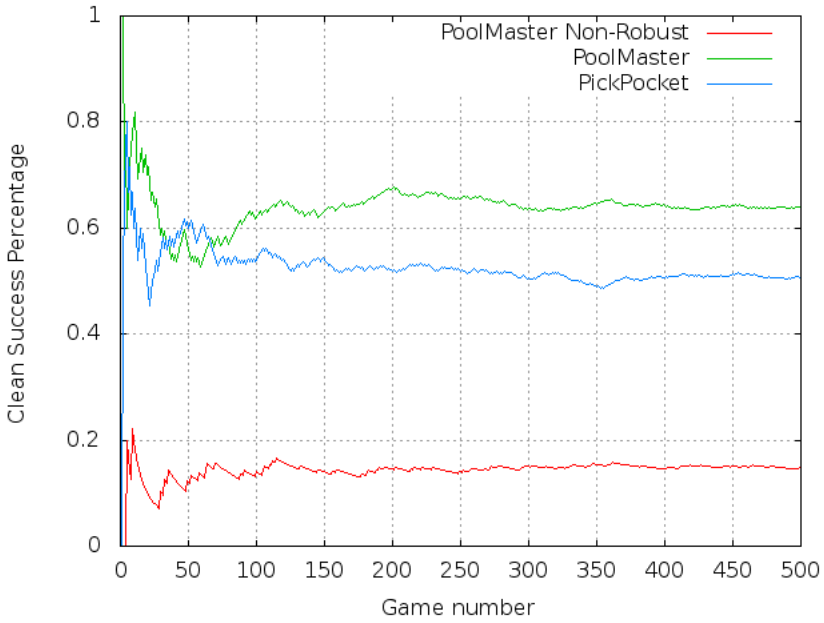

Figure 11: Average rate to clean off the table without missing a shot when a noise factor of 1x is used.

are; a general view of how the player would behave in a tournament when all its features are put to the test.

As can be seen in Figure 12, PoolMaster stabilizes at averages close to 75% over Pickpocket with noise levels of 0 and 0.5x. We could have expected PoolMaster to perform better when no noise was added to the shot parameters, however since both players were using their own break shots, it may have played an important part in keeping the turn after the break. Since the break shots are always different even when no noise is added to the shot parameters, it is not guaranteed that it will succeed every time.

If we now look at the results for a noise level of 1x, the difference between the two players is not as significant, with PoolMaster finishing at an average of 64%. Once again if we refer to [3] where tests were performed against the same player, the margin we have here is not as large (74% reported for Cuecard when using a single-CPU version). However since many other factors come into play when playing full games, such as ball-in-hand behind line (see [7] for detailed rules), safeties and initial choice of breakshot, these will have to be fully tested individually.

Overall, we can infer from these tests that PoolMaster in its current state will have a tendency to perform much better at lower noise levels, also confirming some of the conclusions in [5] that a player doing no planning would be at its advantage in lower-noise situations.

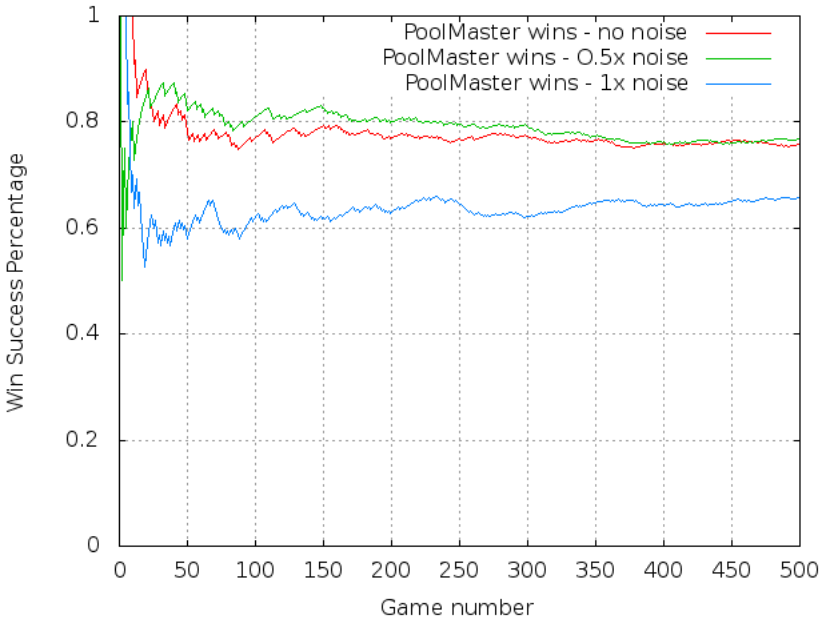

Figure 12: Cumulated average success rate for PoolMaster against Pickpocket, using noise levels of 0, 0.5x and 1x

6. Future work

The aspects discussed in this paper were mainly focused at the controller aspect of our planner. The reason for this choice is that at the moment, 8-ball was the best variant of billiards to test strategies with the upcoming 4th Computational Pool Olympiad. It was also our feeling that even though planning could be helpful in a very small number of cases in 8-ball like shown in figures 3 and 4, given the short time-limits, the advantage was offset by the time required to perform the additional computations to look-ahead while it could be better used to find a better robust shot. It is very likely however that in other variants, planning will be more challenging and require greater analysis, such that the importance of a good planner may be as great as the need for a good controller. For this reason, and to hint onto the direction of future work to complete the two-layered approach, we still provide a base model for the planner.

6.1. Planner

Since we possess a controller on which we can rely to provide us with feasible shots, and a given success rate, we can define a new way of discretizing our search space. If a defines an action on a table state s, and our decision variables defined by:

- b: target ball, from 1 to n,

- p: target pocket, from 1 to 6,
- c: repositioning target,

then it is possible to define the transition function as $s' = f(s, b, p, c)$.

The model defined in 2.3 remains valid with actions u now redefined for each combination of the decisions (b, p, c) for all possible shots. Thus, we find:

$$v'(s) = \begin{cases} \max_a \left[\int_S v(\cdot) dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 1 \\ \min_a \left[\int_S v(\cdot) dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 2 \end{cases}$$

with actions a now redefined. The set of actions A is now partitioned in a subset of $\{b, p, c\}$ combinations for all possible shots. Thus, we find ourselves with a shot sequence represented by:

$$\left\{ \begin{array}{c} b_1 \\ p_1 \\ c_1 = ? \end{array} \right\} \rightarrow \left\{ \begin{array}{c} b_2 \\ p_2 \\ c_2 = ? \end{array} \right\} \rightarrow ... \rightarrow \left\{ \begin{array}{c} b_n \\ p_n \\ c_n = ? \end{array} \right\}$$

where we know the target balls and target pockets, but wish to determine the repositioning targets to reach our objectives on a n shot horizon.

6.1.1. Shot sequence planning

To solve model (2.3), it is first necessary to proceed to a few simplifications. Indeed, it is clearly impossible to do a complete search of the min-max for the game of billiards. If we take the game of chess for example, the complete sequence of shots is directly dependent on our opponent. It means the structure of the game forces the turns to be alternated between both players, and as such it is mandatory to follow a strategy in which we will minimize the possible value of the shot chosen by the opponent. With billiards, however, this kind of situation remains a special case; a player that makes no mistakes will have the possibility to finish the game without his opponent even doing a single shot. Thus there exists strictly two situations in which the adversary will gain his turn:

- A defensive shot.
- A missed shot.

Since the global aspect of the model is already about minimizing the chances of missing a shot, we still need to find a way to assess defensive shots.

A defensive shot, optimally, would normally correspond to a computation by which a player would decide that it is more advantageous to let his opponent play in a bad position, hoping to regain control of the game afterwards, rather than play offensively. Since billiards is a continuous game in nature, the infinity of shots available to the opponent is already hard enough to predict without also trying to predict the case where the opponent would play more than one shot without losing his turn. It is why using this model, we will restrict our search to successful defensive shot to a horizon 1. This means that a defensive

shot leading to a situation where the opponent has any possible shot at all will be defined as a failed defensive shot.

Thus when planning our shot sequence we'll have to deal with the fact that we try to maximise our chances to keep playing until the end, with the special case of possibly playing defensively with a horizon of 1 shot. Assuming that we know the order in which we will execute the shots on the table, then the problem can be resumed as a simple problem of finding the repositioning targets.

If we define the following variables:

- $b(u)$: target ball position after fixing the shot parameters u
- p: Pocket for the given target ball.

If we consider u_n as the shot parameters of our n^{th} shot, we can minimize function f as:

$$\begin{aligned} \min & f(u_0, u_1, ..., u_n) = \text{Posval}(c_0, c_1, ...c_n) \ \forall t : 0...n \\ & \text{s.t. } \|b_t(u_t) - p_t\| = 0, t = 1, ..., n \end{aligned} \tag{6}$$

where the function f would correspond to the position value heuristic described in [18], which associates a value with a position relative to the available shots at this position(depending directly on the position reached c_n).

By using this model, we wish to search for the repositioning targets $c_1, c_2, ...c_n$ to reach after each shot on the table, as can be seen on Figure 13.

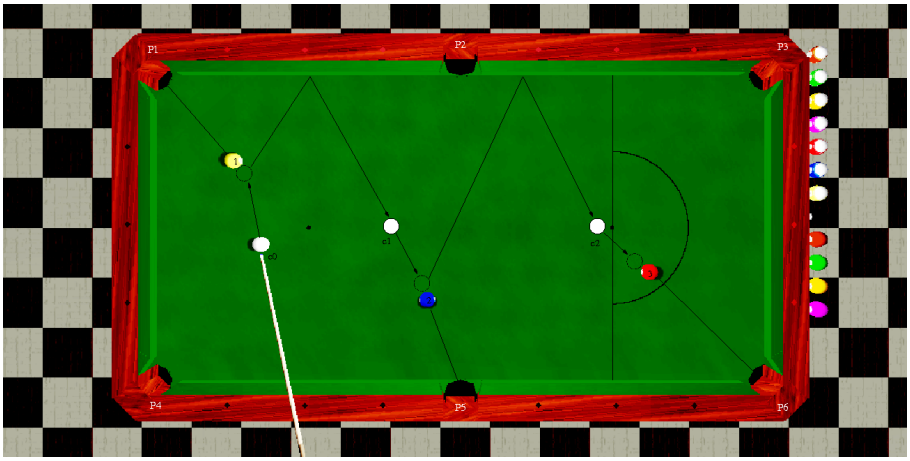

Figure 13: Shows sequence determined by the repositioning targets c_1, c_2, c_3.

Thus, by finding the solution of (6), we will find a list of targets to reach to maximise our chances of shot success. However, we should not forget that this is a very simplified version of the problem. We ignore for the moment the dynamical aspect of billiards, which makes it very hard to define targets $c_1, c_2, ...c_n$ without having an impact on the rest of the system. When optimizing

a shot, it is possible that the cue ball enters in contact with more than one ball, thus changing the global table state and changing the value of the solution we were hoping to find. It illustrates the convenience the robust version of the problem where we aim for a given zone instead of a specific target.

7. Conclusion

The aspects which are described in this paper represent the tools we developed to obtain a fully functional player, with all of the abilities observable in a professional player. We have shown the potential of a using multi-objective model to achieve any desired shots on the table, so that the controller used can easily be applied to any forms of billiards games. We have also shown the effectiveness of using a robust model to only generate shots that will succeed under noise-induced parameters. Our hope is that by carefully using these tools with proper planning, it may be possible to create a decision-making AI able to challenge world-class professional players. A lot of work remains to be done on the analysis of safety shots versus offensive shots, but these are also very much game-dependent, and will need to be adjusted accordingly. The planning aspect for the sequence of shots to be chosen is also likely to be a lot more important when other variants than 8-ball will be played. As our test results showed for 8-ball, a success rate of 100% when no noise is added to the parameters means that it is easily possible to find a solution which leads to victory. Thus it is only a matter of making these solutions robust to noise to create a stronger player. Some situations like the one shown in section 3 will still require planning in the game of 8-ball, but their occurrence seems to be very rare. It also motivates us to explore other games variants in future work like Straight pool, where a player has to call each of his shots, even on the break, making the game planning and multi-objective shots much more important. It will also be interesting to test the approaches described here in a game like Carambole, where less planning is required but better shot exploration is needed, thus possibly taking advantage of the technical skills of the player.

[1] AAAI. Association for the Advancement of Artificial Intelligence. `http://www.aaai.org`.

[2] David Alciatore. *The Illustrated Principles of Pool and Billiards*. Sterling, 2004.

[3] Christopher Archibald, Alon Altman, and Yoav Shoham. Analysis of a winning computational billiards player. In *IJCAI'09: Proceedings of the 21st international joint conference on Artifical intelligence*, pages 1377–1382, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[4] Christopher Archibald, Alon Altman, and Yoav Shoham. Success, strategy and skill: an experimental study. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 1089–1096, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[5] Christopher Archibald and Yoav Shoham. Modeling billiards games. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *AAMAS (1)*, pages 193–199. IFAAMAS, 2009.

[6] Jane Bayes and William Scott. Billiard ball collision experiment. In *Am. Jour. Physics vol. 3 no. 31*, pages 197–200, March 1963.

[7] BCA. Billiard congress of america, May 2008. `http://www.bca-pool.com`.

[8] Aharon Ben-Tal, Stephen Boyd, and Arkadi Nemirovski. Extending scope of robust optimization: Comprehensive robust counterparts of uncertain problems. *Mathematical Programming*, 107:63–89, 2006. 10.1007/s10107-005-0679-z.

[9] Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization – methodology and applications. *Mathematical Programming*, 92:453–480, 2002. 10.1007/s101070100286.

[10] S. C. Chua, E. K. Wong, and V. C. Koo. Performance evaluation of fuzzy-based decision system for pool. *Appl. Soft Comput.*, 7(1):411–424, 2007.

[11] S.C. Chua, E.K. Wong, and V.C. Koo. Pool balls identification and calibration for a pool robot. In *ROVISP 2003: Proc. Intl. Conf. Robotics, Vision, Information and Signal Processing*, pages 312–315, January 2003.

[12] Javier Ruiz del Solar, Eric Chown, and Paul-Gerhard Plöger, editors. *RoboCup 2010: Robot Soccer World Cup XIV [papers from the 14th annual RoboCup International Symposium, Singapore, June 25, 2010]*, volume 6556 of *Lecture Notes in Computer Science*. Springer, 2011.

[13] Jean-Pierre Dussault and Jean-François Landry. Optimization of a billiard player—position play. In van den Herik et al. [28], pages 263–272.

[14] 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):46–53, 2008.

[15] Stanford Computational Billiards Group. Fastfiz billiards simulation engine, 2010. `http://billiards.stanford.edu` .

[16] ICGA. International Computer and Games Association. `http://ilk.uvt.nl/icga/`.

[17] Steven G. Johnson. The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt, 2010.

[18] Jean-François Landry and Jean-Pierre Dussault. AI Optimization of a Billiard Player. *Journal of Intelligent & Robotic Systems*, 50:399–417, 2007. 10.1007/s10846-007-9172-7.

[19] Will Leckie and Michael A. Greenspan. An event-based pool physics simulator. In van den Herik et al. [28], pages 247–262.

[20] Z.M. Lin, J.S. Yang, and C.Y. Yang. Grey decision-making for a billiard robot. In *IEEE Intl. Conf. Systems, Man and Cybernetics*, pages 5350–5355, October 2004.

[21] Wayland C. Marlow. *The Physics of Pocket Billiards*. Marlow Advanced Systems Technologies, 1995.

[22] Thomas Nierhoff, Omiros Kourakos, and Sandra Hirche. Playing pool with a dual-armed robot. In *ICRA*, pages 3445–3446. IEEE, 2011.

[23] Michael James David Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, Cambridge England, 2009.

[24] Ron Shepard. *Amateur Physics for the Amateur Pool Player*. self published, 1997.

[25] Sang William Shu. *Automating Skills Using a Robot Snooker Player*. PhD thesis, Bristol University, 1994.

[26] Michael Smith. Running the table: an AI for computer billiards. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 994–999. AAAI Press, 2006.

[27] Michael Smith. PickPocket: A computer billiards shark. *Artif. Intell.*, 171(16-17):1069–1091, 2007.

[28] H. Jaap van den Herik, Shun chin Hsu, Tsan sheng Hsu, and H. H. L. M. Donkers, editors. *Advances in Computer Games, 11th International Conference, ACG 2005, Taipei, Taiwan, September 6-9, 2005. Revised Papers*, volume 4250 of *Lecture Notes in Computer Science*. Springer, 2006.

[29] J. Walker. The physics of the draw, the follow, and the massé (in billiards and pool). In *Scientific American*, July 1983.

[30] R. Evan Wallace and Michael Schroeder. Analysis of billiard ball collisions in two dimensions. In *Am. Jour. Physics vol. 56 no. 9*, pages 815–819, September 1988.