

Pong: An Introduction to Implementing Computer Game Strategies

Ryan A. Harris and Jayesh B. Gorasia

Abstract—As the modern video game industry continues to grow, more and more people are entering the field of game design. In order to enter this field, prospective game designers must first learn the basics of programming a computer strategy. Here, we follow the development strategies for a basic game, Pong. This development followed a progression from basic to advanced, with the advanced strategies improving upon the basic strategies' flaws. We then pair the basic and advanced strategies against each other and determine the value of our efforts.

I. INTRODUCTION

IN 1972, a bar called Andy Capp's became the first location for a historic step forward in gaming technology. For 25 cents, two players competed in a virtual game of table tennis, using joysticks to move paddles (simulated by lines) up and down the game screen in an attempt to hit a bouncing ball so that their opponent would miss. Atari Inc.'s Pong was not the first video game created, but its simple game play attracted a widespread audience. The morning after the game's introduction at Andy Capp's, a line ran out the door to play this new novelty. By the end of that day, the quarter collecting device had overflowed and jammed the machine. Pong was an undeniable success, becoming the world's most



Figure 1: In the 1970s, Pong changed the world of entertainment as the first widespread console game, merging the popular game of table tennis with television. Despite its simplicity compared to modern console games, Pong was a historic success.

popular arcade game with sales of over 100,000 machines.^{[1][2]}

Pong and similar reproductions soon crossed from arcades into homes in console form. From these curious beginnings, the video game industry quickly erupted and now, just 35 years later, annually generates \$10 billion in revenues. As the industry grows, there are increasing opportunities for passionate gamers to enter careers in game design and development. Schools, such as DeVry University and the University of Advancing Technology, are noticing this movement and offer Bachelor's Degrees in game programming.^[3]

Unlike Pong, which required two players for a game, modern games have single player modes, in which a user plays against a computer. Modern game developers program these computer players to follow strategies, so that they imitate a human player. In learning to develop strategies for a computer player, prospective game designers imitate basic games. Few games are more basic than the first game to reach widespread popularity, Pong.

We therefore intend to study a progression of computer strategies for Pong, from basic to advanced, in order to give prospective game designers insight into the process of developing game strategies. Our study involves determining multiple algorithms for a computer player and implementing them to find the most successful option. This entails a two stage process. First, a Pong implementation is created with a ball that bounces off of walls and moving paddles. Second, we create basic and advanced offensive and defensive strategies for a computer player and test them against each other.

II. IMPLEMENTATION

Although implementing computer strategies is our primary goal, much effort in game design goes toward creating a platform to apply a computer strategy to. Instead of focusing on developing an entire new game, we ~~therefore reproduced~~ reproduce a classic game, allowing us to put more effort on developing strategies. Our Pong platform is written in MATLAB and includes a standard set of rules and parameters. Using knowledge of Pong games from experience with versions found on the internet, we define reasonable parameters for the size of the court and paddles and the speed of the ball and paddles. We also determine a method for moving the ball across the Pong court and bouncing it off walls and paddles. The result is the Pong court shown in Figure 2.

The details of our platform decisions can be found in the

Appendix as we will now focus on our Pong strategies.

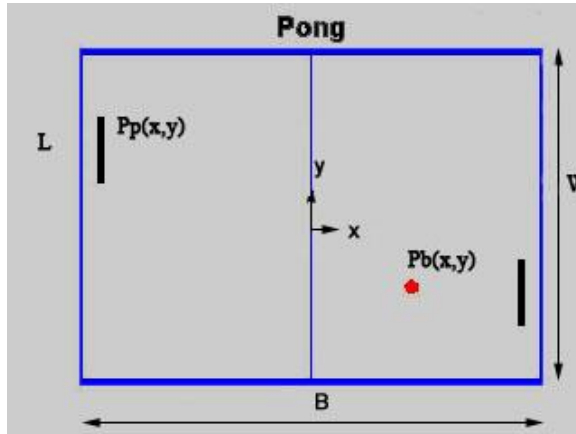


Figure 2: Our Pong implementation created in MATLAB. The labeled parameters are position of paddle, $P_p(x,y)$, position of paddle, $P_b(x,y)$, width of court, W , breadth of court, B and length of paddle, L .

III. DEVISING COMPUTER PLAYER STRATEGIES

In Pong there are two phase of play: offensive and defensive. The offensive phase occurs when the ball is moving towards the paddle; it returns the ball to the competitor's side. The defensive phase occurs when the ball is moving away from the paddle; it is a preparation stage for the competitor's next offensive move. In developing successful strategies, there is a progression from simplicity to complexity, in which the flaws of a simple strategy are overcome by a more intelligent, advanced strategy. For consistency, the ball hitting an upper or lower wall will be referred as a bounce, and the point where the ball would hit a left or right paddle will be referred to as a collision point.

A simple and unbeatable algorithm ~~would~~ will include matching the paddle's position to the ball's position, for both offense and defense. However designing an unbeatable game is not lucrative for a game designer as human players find no satisfaction in trying to achieve impossible a challenging victory. By limiting paddle speed, the paddle cannot keep pace with a bouncing ball and the unbeatable algorithm described above is not viable will not be feasible. The challenge, then, of designing a Pong computer strategy is to create an effective method for paddle movement with limited paddle speed.

A prospective game designer should focus on basic computer strategies until their skills are have improved to handle more advanced problems. In some Pong games, moving the paddle as the ball strikes it affects the reflected angle of the ball. As a simplification, our version of Pong does not consider this aspect and bases the reflected angle of the ball only on the position of the collision between ball and paddle, as explained in the Appendix. Therefore, our strategies only consider the position of the paddle with respect to the ball and not the velocity of the paddle.

A. Offensive Strategy

Initially, we employ a simple offensive strategy. This is an adjustment of the unbeatable method described earlier to include limited paddle speed. If the vertical position of the

ball is above the paddle, the paddle moves up; if the vertical position of the ball is below the paddle, the paddle moves down. Unfortunately the paddle speed is too slow to account for bounces with the upper and lower wall. The failure of this strategy is evidence that a method for predicting the collision location is necessary for a successful offensive strategy. The paddle should move to the collision location and wait for the ball instead of being influenced by the ball's bouncing.

An algorithm to predict the location of the collision proves more successful at consistently hitting a bouncing ball and is used as the general offensive strategy. We will refer to this algorithm as the basic strategy. Regardless of the position or velocity of the ball after the collision with the opposing paddle, the same algorithm applies. An effective computer strategy must work for all scenarios or else it is flawed. The algorithm uses the horizontal velocity, vertical velocity, and length of the court to determine the total vertical distance traveled by the ball. This value is then adjusted for bouncing by removing multiples of the court width to determine the final vertical position of the collision. The system of adjusting for bounces is demonstrated in figure-Figure 3.

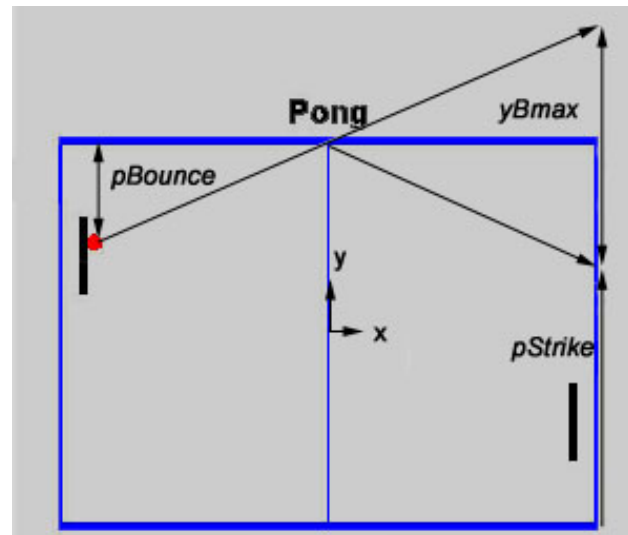


Figure 3: Above is a visual representation of the bounce adjustment calculation for our final offensive strategy. Using the horizontal velocity of the ball, the time required to cross the court is found. This time is used to find the vertical distance traveled by the ball (y_{bmax}). If y_{bmax} is greater than the distance between the collision point and the wall ($P_{Bounce1}$), the difference between y_{bmax} and $P_{Bounce1}$ is removed from y_{bmax} and reflected to represent a bounce. These steps are repeated until y_{bmax} is reduced to a position point within W . The result is P_{strike} , the position of the ball at the collision point.

A major flaw of the basic strategy is that it does not attempt to outplay the opponent; it simply moves the paddle to a location so that it can hit the ball. To improve upon the basic strategy, we created an advanced strategy, which extends the basic offensive strategy to include aiming. As part of the Pong platform, the position of the collision on the paddle affects the angle at which the ball bounces off of the paddle, a feature described in detail in the Appendix. Based on the position of the ball on the paddle when they collide, there is a limited range of locations the ball can return to reach on the opposite side. A uniformly distributed sampling of points is taken throughout this range. The aiming algorithm then calculates which point would be most difficult for the

opposing paddle to reach, based on the velocity, V_{req} , that that the opposing paddle would require to reach the collision point. The algorithm hits the ball to the point that maximizes paddle velocity determined by the equations:

$$t = \frac{B}{v_x} \quad (1)$$

$$V_{req} = \frac{P_p - P_{strike}}{t} \quad (2)$$

Increasing the number of points in the sampling increases the accuracy of the aiming method, but increases computation time, a tradeoff which is analyzed and compromised by the programmer. Although a game as simple as Pong does not require a lot of processing, advanced video games are far more complex, and the ability of modern consoles to meet the requirements of a video game is an important consideration for a game designer. If the computer or console does not have enough processing speed to handle a video game, the video game will ~~constantly freeze~~ not play smoothly, reducing the entertainment value of the game for a human user.

B. Defensive Strategy

The basic defensive strategy moves the paddle to the center of its side of the court. This method is a step above not moving the paddle after a collision because the paddle is left in a position to return a maximum range of balls. However, a ball headed to a corner of the court is difficult to reach because of the limited paddle speed.

The major flaw of a basic centering strategy is it does not react to the opposition and leaves the corners open for an aiming strategy. The advanced strategy uses a similar technique to the advanced offensive strategy in order to predict the optimal place to locate the paddle to minimize paddle movement. The sampling of the range of locations that the opponent could possibly hit the ball to is taken and the paddle moves to the center of that range.

This strategy gives the advanced paddle a significant advantage over the centering strategy, as the paddle does not have to travel as far when it correctly predicts the collision point. Therefore, limited paddle speed becomes less of a factor. Similar to the advanced offensive strategy, there is a limitation on the number of points in the range that can be tested because of computational speed.

C. Results of Basic vs. Advanced Pong Strategies

In competition, the basic strategy calculates the collision position of the ball offensively and moves the paddle to center defensively. The advanced strategy uses the same offensive approximation method but adds aiming to hit the ball to the point most difficult for the basic paddle to reach. Defensively, the advanced paddle determines an optimal resting location to prepare for the next offensive phase. One thousand trials between the two computer strategies result in 904 wins for the opposing team, confirming that the effort to improve upon the basic strategy to negate its flaws is worthwhile.

A typical round involves fourteen to eighteen successful volleys, with the advanced paddle hitting the ball to the basic paddle's corners. The basic paddle moves up and down the court to return to its centering position. As the ball speeds up with each paddle strike, described in the Appendix, it becomes more difficult for the basic paddle to reach the ball until an eventual loss. When the basic paddle wins, there are typically fewer volleys, four to eight, as the advanced paddle's defensive strategy incorrectly predicts the collision point of the ball.

IV. CONCLUSION

In progressing through strategizing, we initially made basic ~~ones~~ strategies for offense and defense and determined the flaws associated with ~~these strategies~~ them. We then improved upon these flaws by extending our basic strategies into more advanced algorithms, which actively determined optimal moves. The advanced strategy was a large step forward, losing only in the rare case its defensive prediction method made an incorrect prediction. This method of improving upon flaws applies when designing any game; ~~we used~~ Pong ~~is~~ only ~~as~~ an introductory example.

We had hoped to implement other conditions on the pong court and move past the basic game that has been played for over 35 years, but because of time constraints we did not have the opportunity to experiment with these extensions. Our strategies could have then been put through far more rigorous testing and might have required a complete redesign. Some examples include,

- Ball spin can be included based on the velocity of the paddle as it collides with the ball. Strategies would include limiting speed so the paddle hits the ball while moving, instead of focusing only on moving the paddle to a position.
- A third dimension of motion can be added to the playing field. A strategy would involve moving the paddle in two dimensions and there would be challenges in animation as the ball moves ~~in a direction into and out through the plane~~ of the computer screen.
- Obstacles could be placed randomly throughout the court. A proper strategy would aim the ball away from the obstacles to prevent bounces back, but as the obstacles would be randomly placed, the strategy must be flexible.

Having experienced the progression of Pong strategizing, we feel pursuing such extensions would be a worthwhile challenge for a prospective game designer.

APPENDIX-PONG PLATFORM

Before developing strategies, a game designer must create a platform for the game. Our Pong platform is written in MATLAB and includes parameters for court size and game speed, a method for moving the ball across the court, and a method to control collisions with the paddle.

The horizontal length of the court is double the vertical

length of the court so each side is a square. The paddles are one fifth the size of the vertical height. The limit on the paddle speed is one fourth the initial ball speed. This limit ensured that the paddle could not travel from one side of the court to the other in the time the ball crosses the court. The small paddle size and the limit on paddle speed combine to increase the challenge of creating a strategy for a computer player.

As the ball travels across the court it moves with constant velocity just as in the original Pong game. The ball then bounces off walls or collides with paddles. We devise two methods for this ball movement, so we can explore the positives and negatives of each and choose the optimal design for implementing computer paddle movement strategies.

- Euler's method is the first examined, employing a constant time step and the velocity of the ball to determine the next position of the ball. When the position of the ball crosses a bound, the walls of the court, the velocity component is negated, and the ball position is adjusted for time step error. Because of the time step, the ball will travel slightly further than the wall and must be placed back on its original path.
- The second method examined uses a differential equations solver, ODE45, with an events option. The differential equations solver produces a set of points, which the ball moves through. The solver cannot produce an instantaneous force on the ball, which would represent a collision. This is overcome by stopping the solver at every collision, using the events option. Another instance of the solver is then run with a component of the velocity negated, up to the next collision.

Euler's method requires a complicated script and only is applicable if the ball moves in a straight line. However, since it is constantly predicting the ball's next position, the animation of Pong is smooth. The ODE45 method is simpler, as its code is only half as long as the Euler's method's [code](#) and it can be extended to include forces on the ball. Since it calculates a new path for the ball at every collision though, the animation stops at each collision until the new path is calculated. Euler's method is the preferred method in our situation because of the smoother animation, which is more similar to the original Pong game, and because time constraints were too tight to allow for extending the project to include forces, such as gravity or spin, on the ball. Also, the point by point movement of Euler's method is useful in obtaining coordinates of the ball on the court for calculations.

Although the bounces off walls and off paddles are produced with the same Euler's method, there are differences between the two types of bounces to improve game play. Bounces with the upper and lower walls are elastic; the vertical component of velocity is reversed, while the horizontal component remains constant. Collisions with paddles are not elastic. To ensure that one side will eventually win, the vertical component of the velocity is increased by two percent every time the ball hits a paddle. With a limited

paddle speed based on the initial velocity of the ball, the ball eventually travels too fast for the paddle to reach it. The paddle also changes the angle of reflection of the ball as it hits. The amount of this change is proportional, up to a maximum, to the distance from the center of the paddle. Where θ is the angle of incident and θ_n is the angle of reflection, the proportional change follows the equation:

$$\theta_n = \theta + \frac{\pi}{4} \frac{y_b - y_p}{\frac{1}{2}L} \quad (3)$$

The effect of this equation is demonstrated in Figure 4.

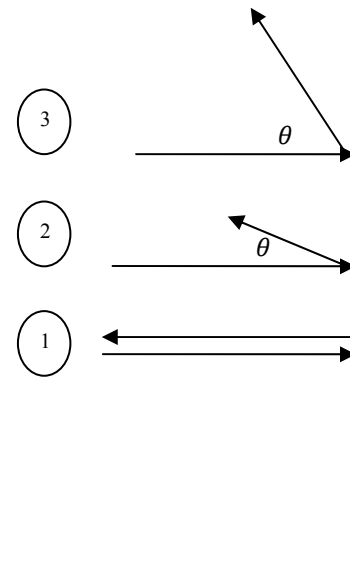


Figure 4. As the ball hits the paddle, it is reflected at a greater angle based on distance from the center of the paddle. Demonstrated by three theoretical situations, if the ball hits the center of the paddle, it is reflected at the same angles before, and the angle increases up to a maximum as the ball strikes closer to the paddle edge. The same progression of larger angles occurs moving downward from the center. θ is the incident angle of the collision.

In game play the ball's starting position alternates sides and moves at a random angle towards the opposing paddle. This measure ensures that both the basic and advanced strategies are tested to fair and reasonable conditions.

All of these choices keep in mind the original Pong game and the limits of our programming skills, and successfully forms a platform for implementing Pong strategies.

REFERENCES

- [1] "Pong." - Wikipedia, the free encyclopedia 3 February 2008, - <http://en.wikipedia.org/wiki/Pong>
- [2] Winter, David. "PONG-Story." 29 February 2007, - <http://www.pong-story.com>
- [3] "About Game and Simulation Programming" DeVry University. http://www.devry.edu/programs/game_and_simulation_programming/about.jsp