City College of New York

Pong 1001

Jezlea Ortega, Carlos Rodriguez

Software Design Lab 22100

Professor Albi Arapi

December 7, 2023

Introduction:

Pong 1001 (2023) is a modernized developed sequel of Atari's groundbreaking hit *Pong* (1972). The original *Pong* was first released in 1972 and served as an alternative version to the typical table tennis game known as *Ping Pong*. Atari's version of *Ping Pong* was an arcade game initially until exponential sales prompted the development of *Pong* into a console system game in 1975. However, despite its grand success, the game ultimately lost popularity by the 1980s due to a lack of focus on video games during that period. In *Pong 1001* (2023), we are bringing back the classic Atari game with some twists of in-game mechanics to fit the more current video game player. Originally, the concept was inspired by a recent trend in *Tetris 99* (2019), an extension of the original *Tetris* (1984) but with 100 players battling against each other. The introduction of such game mechanics provided players with fresh challenges in a highly competitive environment and a new experience. The creation of *Tetris 99* became widely popular as it took an originally groundbreaking game and redesigned it to meet the current type of gamers. The apparent trends related to retro gaming and the desire to experience their nostalgia provide the need for designing a game such as *Pong 1001* (2023). Through the creation of *Pong 1001* (2023), we aim to achieve a similar effect as *Tetris 99*, taking *Pong* (1972) and altering its game mechanics by including a ball-splitting feature to appeal to current video game audiences while commemorating the original hit.

Objective:

The objective of *Pong 1001* (2023) was to recreate the original *Pong* (1972), with new game mechanics and designs. Through such changes, we would provide the player with grander challenges, an increase in entertainment levels, and a greater potential for strategy, while paying

homage to the 1972 version. In the creation, we implemented an overhaul in the original mechanic of simply battling against another player and the speed of a pong ball increasing with time. Our alternative version instead places opponents in a 1V1 battle where a new ball is added every thirty seconds. The number of balls continuously increments until one player wins, each maintaining a hundred lives to accommodate this new level of difficulty. The balls have physics and may sometimes roll, slow down, or speed up depending on how they collide with the players or walls. This opens up opportunities for players to develop strategies against one another. This extensive change will challenge players to utilize strategizing skills, from deciding the method at which they will defend or choosing to focus on spiking a pong ball. Ultimately, the increased difficulty will provide higher entertainment levels between duos and resurge players' love for *Pong.*

Title Screen:

The first step we took when designing the "Title Screen" scene was to create a new scene. We simply made the background black on the main camera's color wheel, added music to it, and then began by adding a new TextMeshPro to our canvas. This TextMeshPro maintained our game title, *Pong 1001*. However, when designing our logo we did not write the number "1001" and instead wrote "00". We did this to have the player paddles serve as the 1s for aesthetic purposes. Ultimately these additions are made with a desire to pay homage to traditional retro games, whilst implementing a fresh 2023 visual appearance and retaining the minimalism of Pong. We had to create a "Change To Game" script and add it to the scene to implement keyboard features that would allow the game to begin or close. Through the utilization of Unity's Input.GetKey we encoded the Title Screen to load the game screen if "space" is pressed or close if "Q" is

employed. Additionally, utilizing Input.GetKey once again we encoded the control scene to open upon “C”. The finishing feature of our title screen is the addition of three TextMeshPros with animations indicating to the user which buttons must be pressed to either begin *Pong 1001,* open the controls screen or close the application.

Controls Scene:

Every game developed in recent years has seen many different control schemes and layouts. Most games nowadays have the option to use keyboard and mouse, controller, and even Virtual Reality Devices as well. Our game is simple with its controls because of the game it is based off of. We are not moving in a 3D environment so the movements are constrained to up and down for both players 1 and 2. The controls themselves were easy to implement, but to assist users that might not be too familiar with conventional game controls, we have added a control scene to specifically list out keys for each player as well as the goals of the game.

Pong Game Screen:

Identical to the title screen’s first step, we started by creating a new scene called “Pong” and setting it to layer one. This particular scene was also set to a black background and given music in its main camera portion. The “Pong” scene housed all our game objects necessary to implement the gameplay and mechanics for our players. Further information regarding the creation of our game objects can be found in the proceeding sections of writing: player paddle, ball copies, game manager, etc. In regards to the design aspect, we utilized three square game objects to add additional lines to our background. The lines were set to white in their color portion followed by a resizing to 0.01 for their x and 10.23 for their y. Utilization of resizing

allowed us to create thin, long lines, to fit the canvas size and act as somewhat of a border for each player's respective side.

Walls:

The walls of Pong 1001 were fairly easy to design. The way the original game worked is that the ball could bounce in between 2 walls: The top and bottom walls. The left and right walls of Pong served as scoring zones. When the ball enters the scoring zone it resets to the middle and gets sent in a random direction. For Pong 1001, every wall had a collision, and while there were scoring zones, the balls were not reset upon colliding with the walls. The walls were given collision by built-in Unity objects. We made game objects and attached 2D Box Colliders to them to give them collision properties automatically handled by Unity. We set up the colliders in the shape of a rectangle to hold our entire game environment.

Paddle:

The paddles of each player are crucial in the game of Pong. Each paddle is a simple rectangle with collisions and physics attached to them with the addition of movement options. These paddles bounce the balls back and forth between the game screen and allow the game to be played. In order to design these paddles we first had to make rectangle objects in the Unity gamespace and attach colliders to them. Once we wrote a simple script for each of the paddles, they could now move and collide with the walls of the gamespace as well as the balls. Our original color scheme was supposed to emulate the original Pong, but we eventually decided upon using a red and blue scheme throughout in the hopes of the players being able to easily identify themselves.

Ball & Copies:

For the ball object we first needed to create an object in the Unity gamespace. We decided on a ball instead of the original small square used in Pong. We gave our ball a circular collider as well as a rigidbody object so that it would have physics in a 2D environment. Once the colliders were set up for the ball, we needed to attach a script to it so upon collision with one of the goals, the balls would trigger a scoring system and update the game. Since our ball object was now set up, it was very easy to instantiate multiple copies. First and foremost, to implement the Ball Copies Game object, we began by duplicating the original game object, “Ball”, and renaming the duplication to “BallCopy”. Through such duplication, we maintained a game object that held an identical RigidBody and script to the original object, thus holding its properties and code. After this creation, we dragged the “BallCopy” game object to the assets section to create a prefab. Our prefab of “BallCopy” would now serve as our blueprint, so we could delete the “BallCopy” game object. Next, we created a new game object called “Ball Copies”, which would maintain a new script “BallDup” that codes in a duplication method for our pong balls. In “BallDup”, we defined three public variables; time (keeps track of current time), copy (will hold our prefab game object), and spawn (controls spawn rate). Momentarily, we left our “BallDup” script, saved our code, and dragged our prefab to the copy variable section. Given our variable definitions, in the update portion, we indicated that if time is greater than spawn we know a new ball should spawn, and as such make a copy. Using Unity’s duplication feature, instantiate, we simply indicated we wanted to instantiate copy, transform position (keep the x and y coordinates identical), and transform rotation (keep the rotation identical). Given our prefab maintained code identical to our ball class it would follow the x and y coordinates indicated within our “Ball” script. In this if

statement, our final piece was to reset time by defining it as zero. Now, we defined an else portion that indicates time is not greater than spawn, and as such we wish to increment time. Through the utilization of time and spawn, we created a controlled spawn rate of copies, thus preventing crashes and repetitive duplication every second.

Game Manager:

Furthermore, we now created a new game object, “Game Manager”, with a script called “Game Manager” that would store our score system alongside functions for each player. Before implementing our Game Manager script, we created four TextMeshPro’s on our canvas. Two TextMeshPros would initially represent each player's score, residing as “Score: 0” until either player scores. Similarly, the other two TextMeshPros would label the players’ lives, residing as “Lives”. In the Game Manager script, we defined four private int variables, each player having an int variable storing lives left and one storing their score. Similarly, we defined four private TextMeshProGUI’s maintaining the displaying text for each player's lives and scores respectively. In regards to the defined variables, our final step was to define two Images, each one storing the image representing players’ HP bars. Before continuing, we saved the code and returned to Unity, placing the TextMeshPro associated with each private TextMeshProGUI variable in their slot and Images within their respective section. As we returned to the script, we implemented two functions for each player. In player one’s function, we indicate if a ball enters its respective goal, a life is lost so player one’s HP bar fill amount decreased by 0.01, and player two’s score variable increments. As such take the score text associated with player two and rewrite it as “Score:” plus the variable tracking their score. Utilizing an if statement, we indicated if the player’s life equals zero the individual had lost, and thus take them to player

one's game over scene. Concerning player two's function we followed an identical format, subtracting a life from their HP bar and incrementing player one's score if a ball enters their goal. Then, we took the text for player one's score and rewrote it with the incremented score indicating a goal. Finally, using an if statement once again, if player two's lives equals zero it indicates a loss, and thus takes the player's to player two's game over scene.

Player 2 Paddle:

The implementation for our second-player paddle involved creating a new game object, "Player 2 Paddle", and adding a script named "Computer Paddle" for our coding aspect. In our script, we implemented Unity's Input.GetKey so that if a user presses the up arrow versus the down arrow the paddle would move in a particular direction. The code indicated if the key code equals the up arrow, the direction would be upwards and as such add force to that direction. Similarly, if the key code represented the down arrow, the player wished to move their paddle downwards so set the direction downwards and add force. In each respective case, the addition of force would equal our direction times the speed at which the paddle is set. Lastly, if neither key is pressed by the player this indicates the user wanted to remain stationary, and as such our paddle would move in no particular direction.

Game Over Screens:

The construction of our game over screens shared many similarities to the title screen. Firstly, we create two new scenes, one serving as the game over for player one, and the other for player two. In each scene accordingly, we followed the same layout. We created a new TextMeshPro with game-over written inside to indicate an individual has beaten another player. Additionally, we

added another TextMeshPro with animations indicating to the user if “escape” is pressed they may continue. To implement this continue feature we attached our “change to title script” which was also previously used for our control scene. Attaching this script allowed for the title screen to load if the player presses “escape”. Furthermore, we also implemented visual differences depending on which player wins. If player one wins we attached a TextMeshPro to display “Player 1 Wins”, and created a new game object with a square script in their paddle color. Additionally, by focusing the main camera on our paddle color we split our game over scenes down the middle, showing the player’s paddle color during their victory. Dissimilarly, if player two wins we attached a TextMeshPro to display “Player 2 Wins”, and created a new game object with a square script in their paddle color. Once again, by focusing the main camera on our paddle color we split our game over scenes down the middle, showing the player’s paddle color during their victory.

Conclusion:

In the culmination of this project, we successfully created an homage to *Pong* while providing a plethora of new mechanics: ball splitting, the potential for spiking, increased lives, etc. Implementing these changes provided greater potential for strategy building, increased entertainment value, and higher difficulty. *Pong 1001* ultimately served to be a tremendous learning experience for both of us. Heading into development we each maintained a different skillset but shared a strong desire to pursue the field of game development, due to our love of video games. As one team member maintained experience in game development class, and another was simply a beginner, it made for a tremendous pairing. We worked well as a pair, learning to navigate Unity through the utilization of their manual, each other, and websites such

as StackOverflow. Another invaluable learning experience was the process of designing and implementing a game: constructing an idea, visual choices, gameplay choices, game mechanics necessary/desired, debugging, etc. However, speaking to the downsides, we also discovered Unity could be very difficult for collaboration. Due to Unity's collaboration requiring a pay-by-minute fee, it made it virtually impossible for the team to work in unison. Oftentimes, we had to construct code together one at a time and although we fit well together as a pair, it made coding somewhat inefficient alongside time-consuming. We did this by utilizing another collaboration platform called Zoom, which granted us some relief while coding, but was also limited on time due to being a free version. Given this tremendous downside regarding Unity, it sparked a curiosity in us to learn Unreal Engine and if this large issue may be resolved by utilizing a different game design engine. *Pong 1001* had a large variety of features that may still be implemented in the future. As a team, we oftentimes discussed new features such as a backend portion maintaining a database that would store a player's high score, and potentially a leaderboard. Additionally, we discussed in depth in the future implementing a feature to allow players to decide whether to verse an AI or another competitor.