Brozon

Trevor D. Brown and Travis Anderson

Introduction of *Brozon*

Brozon is a *Space Invaders*-type clone, where a player attempts to hit a moving target using a ball of energy (hereafter referred to as a projectile). While attempting to hit the target, a second player, which may be a computer player, tries to block the projectile from hitting the target. While there is no time limit to the game, player one has only twenty projectiles to fire at the target. After all projectiles have been fired, the game is over.

Project Analysis and Design

Brozon is written in Visual Basic. This language was selected because of how easy it was to manipulate objects on screen to perform animations and other various tasks. Visual Basic made it easy to generate graphical elements by simply dragging and dropping the elements to the "Form" - the window currently being edited.

The first challenge was getting a target to display and move. Because of prior experience with Visual Basic, it was already known how to easily add elements to the form, and how to manipulate their positions accordingly. Therefore, a PictureBox object was added to the form, which contained the target. Then, using code ran at runtime, the target was positioned at its appropriate Y coordinate. Then, a timer, tmrTargetMovement, was used to calculate the current position of the target and adjust its horizontal X movement. If the target was visible within the playing field, it was to continue moving to the left. If the target was hidden on the left side of the screen, it would reset to the right side of the screen, remaining hidden. Then, it would proceed to the left again.

The next challenge was getting a line to follow the cursor. The line was to be used to help player one determine the direction of the projectile and the amount of power behind the movement of the projectile. A special "Power Pack" - a framework extension - for Visual Basic had to be installed on the development machine to allow objects such as lines, ovals, and other shapes, to be drawn to the screen. To limit the space a user could use to fire a projectile, an oval object was placed at the bottom of the screen.

From there, using System.Drawing.Point - a class within Visual Basic - start and end points for a line could be easily established. Furthermore, using the autogenerated MouseMove event handler, the end point of the line could be constantly redrawn to match the cursor position. To calculate the power behind the shot, the generated line's rise and run were used to determine how far the projectile would move using timer ticks and its current position. A problem with doing this method is the numbers representing the rise and run of the slope were so large relative to the size of the form, it was almost impossible to see the movement of the projectile. This issue was resolved by scaling down the rise and run to 3.5% of their original values.

To represent power in a visual way, the distance formula of the end points of the line were used. If the distance between the start point and the cursor was very short, the shot was slower.. The larger the distance, the faster the shot. The line turned green if 50% or less of the potential maximum power was used. The line turned yellow if the 50% to 80% of the potential maximum power was used. The line turned red if over 80% of the potential maximum power was used.

The next challenge was getting a projectile to fire on the click of a mouse in the direction of the line drawn. To achieve this, when the MouseDown event handler is triggered, the slope of the current line drawn is calculated using the slope formula, ([y2-y1]/[x2-x1]) where (x1,y1) is the base, starting endpoint of the drawn line and (x2,y2) is the line's endpoint at the cursor's position. After this event is triggered, a timer, tmrUpdate, updates the position of the projectile by adding or subtracting the position of the left and top edges accordingly. Because the top-left corner of the form is position 0,0 on the "grid", movements to the top are always subtracting, while movements to the left and right are subtracted and added, respectively.

The next challenge was determining when a projectile impacts the target. Because all moving objects during gameplay are PictureBox objects, it was easy to determine if two PictureBoxes, namely the projectile, and another object in the playing field, overlapped. If overlap occurred from anywhere on the projectile image, and from only the bottom of the other objects, it was determined to be a hit, and the projectile reset to its original position. The reason it was chosen to be this way was because the other objects have arms, and can only intercept the projectile with their arms. Therefore, the bottom of the PictureBoxes of these images was the ideal place for the collision detection to occur.

The next challenge was scoring. To achieve this, two variables were set up to represent the score of each player. Player One had a dedicated variable. Player Two and Computer shared the same variable, since there were no conditions where Player Two and Computer play at the same time. This was done to also save space. The scoring metric was determined to be 100 points for player one, if they hit the target. If player one hits player two, player two gets 100 points, while player one's score remains unchanged. After all projectiles are shot, the player with the highest score wins the game. These points values were chosen because of their simplicity to manipulate, and because they were fair for the conditions required to score points.

The last challenge was determining player modes. Because of the way our game was designed, it was easy to incorporate a computer player, in case another human player was not available. This is done through boolean values and radio buttons. If Player vs. Player mode is selected, the appropriate boolean value is assigned to bolPlayerTwoComputer, and the game treats player two as a human player, requiring them to control their object on screen. If Player vs. Computer mode is selected, the appropriate boolean value is assigned to bolPlayerTwoComputer, and player two's object is positioned in a randomized location within a certain set of constraints, displayed for a few seconds, hidden and replaced outside of the game board, and repositioned at a randomized location. The constraints are the left and right edges of the form, a few pixels below the bottom of the moving target's PictureBox, and a few pixels above the top of the oval at the bottom of the form. This provides adequate space for player two's object to move, regardless of who moves the object. Additionally, if player two is human, the program is written in such a way as to allow the object to "loop" across the screen. For example, if player two moves the object too far to the left side of the screen, it will move to the right side, and continue going left.

Brozon
Trevor D. Brown and Travis Anderson

Key Features of *Brozon*

- Player One can shoot a projectile in motion from an angle based on the position of the mouse cursor relative to the base of the game window.
- Player One can aim their shot, with it's projected motion represented by a colored line.
- The projectile's speed/power can be adjusted by Player One by adjusting the length of the aiming line. The appropriate power level (low, medium, high) is represented via different colors (green, yellow, red).
- A moving target is present at the top of the screen, moving horizontally to the left continuously, for Player One to shoot a projectile towards.
- An enemy obstacle attempts to block Player One's shots - either by being positioned in a random location calculated by the computer, or controlled by Player Two.
- Both Player One and Player Two can score points based on their performance. Player One scores 100 points for hitting a target, while Player Two scores 100 points by blocking Player One's shots.
- Player One's "cannon" has a limited amount of ammo and chances to score points (as does Player Two). After 20 shots have been fired, the game is over and the Player with the most points wins.
- Alternative controls for Player Two are available: the enemy obstacle can be controlled with either WASD keys or the directional arrow keys.

Usage Instruction

To play the game, extract the contents of the ZIP folder to a storage place of your choice. Please note, the game will not function if any files within this folder are deleted or moved. Please keep all game application files together in the same folder.

When the game application is loaded, the user is greeted with multiple options:

- Start Game
- About
- Quit
- Player Mode options
- Arrow keys options

To start the game, the user must first select their desired player mode and control options. Under "Player Mode" selecting the "Player vs Player" option will have Player One control the aiming cursor and firing of the projectile cannon while Player Two will control the enemy obstacle attempting to block the projectiles. The user may opt to use alternative controls for Player Two (WASD keys instead or arrows) Selecting the "Player vs Computer" option will Player One control the same, while the enemy obstacle will instead be played by a computer that randomly generates its position on the screen. After selecting their player mode and control options, the user can click on "Start Game" to load the gameplay screen. From here, if the Player vs Player mode was selected, Player One controls the aim of the cannon with the

mouse. Using the mouse to reduce and increase the length of the aiming line will determine the power of the shot (with a percentage of said power displayed in the bottom left). The aiming line will change color to represent the power level as well. To fire, Player One must click within the half-circle shooting area. While Player one attempts to shoot and hit the target, Player Two can move the enemy obstacle with the arrow keys (or WASD if the control options were changed) to attempt to block the shots. If either Player One hits the target, or Player Two blocks a shot, then 100 points are awarded to each respectively. The game is over when all 20 of the cannon's ammo is depleted and the player with the most points wins. Selecting the "About" option will bring up the names and photos of the game's creators, with links to their own websites. The "Quit" option closes the game window.

Problems, Bugs, Unimplemented Features

The only known problems with the application occurs when the user attempts to play the "Player vs Player" mode with a laptop trackpad for the mouse. Since this mode requires the enemy obstacle to be moved with the keyboard, both mouse and key input is being processed. With a trackpad, however, Player One will not be able to move the mouse if any key is held down at the same time, essentially making the PVP mode unplayable. This is likely due to both the trackpad and keys sharing an import bus, causing interference with the game's event handling. Another possible cause could be a feature of Windows, where the trackpad is disabled when typing to avoid accidentally moving the mouse cursor and causing errors.

The other problem occurs when player two holds down the Spacebar and attempts to move Up and Left while playing. For some unknown reason, the player two object will not move up and left while holding the spacebar. Otherwise, player two will move up and left without any issues.

References

- https://msdn.microsoft.com/en-us/library/sh9ywfdk.aspx - Visual Basic Language Reference
- http://www.piskelapp.com/ - Used to create cannonball, cannon, enemy target, and friendly target graphics
- http://www.8bitphotos.com/ebpheliohost/dispatch.wsgi - Used to create 8 bit versions of the photos of Trevor Brown and Travis Anderson.