

Breakout in Real-Time

By: Tim Allgire

ECE 4220

Final Project

December 2011

Abstract:

This project is the creation of the popular game Breakout on the TS-7250 board. Breakout is played by a player moving a paddle at the bottom of the screen so that a ball that bounces off of the paddle hits all of the target bars at the top of the screen. The game is implemented using multiple threads and real-time tasks and semaphores are used to ensure that variables that are changed or checked by multiple tasks are not accessed at the same time by different tasks.

Introduction:

For this project, I decided to create an implementation of the game Breakout that uses different concepts that were covered in the class. In the game Breakout, there are a number of targets which are shaped like blocks at the top of a playing area. The playing area is surrounded by a visible and set boundary on the top and sides of the playing area. A ball bounces off of the boundary of the playing area and must bounce off of the targets in order to clear them. The player of the game tries to get the ball to bounce off of the targets by moving a paddle left and right and bounces the ball off of the paddle. The player can help to direct the ball by moving the paddle while the ball is hitting it in order to add “spin” to the ball and change the horizontal velocity of the ball. The player must also keep the ball from falling to the bottom of the playing area by bouncing the ball off of the paddle and if the ball does fall to the bottom of the playing area then the player loses a life.

Background:

The original Breakout game was created in 1976 by Atari and was an arcade console game. Many remakes or different versions of the game have been created since the original game was released, but all the games have the same basic requirements in order to create the game. In order to implement this game, a person has to figure out a way to create a ball and keep track of the position of the ball and the velocity of the ball. The ball must bounce off the targets, the boundary, and the paddle, so the program must know when the ball is going to collide with other objects and be able to adjust the velocity of the ball accordingly. The ball must get rid of the targets when the ball bounces off of them, so the program must be able to remove the targets from the game when the ball hits a target so that the ball can move through the now freed space. The program must also restart the ball and paddle and remove one of the player’s lives when the ball gets below the paddle. The program must take in a user input in order to move the paddle left or right while making sure that the paddle doesn’t move outside of the boundaries of the game. The program must be able to change the horizontal velocity of the ball if the paddle is moving when the ball hits the paddle. The final requirement of the program is that the program must display these things to the screen and must display quickly enough and regular enough that the ball and paddle don’t seem to stutter in their movements.

Proposed Implementation:

The game is a single program that runs in user space. In order to create the game, the program creates multiple threads and then creates a real time task for each thread so that there are three tasks running at the same time. One task takes user input through the buttons and moves the paddle. The second task moves the ball and checks when the targets are hit. The third task displays the game on the screen. Each of these tasks share memory that contains the information for the ball, the information for the targets, the location of the paddle, and two values for when the paddle moves. The information for the ball is in a struct that contains values for the row and column location of the ball and values relating to the horizontal and vertical velocities of the ball. The information for the targets is in an array of block structs. The block struct has a value that is 1 if the block has not been hit and the value is 0 if the block has been hit. There is a variable that contains information for the color of the block in the struct. The other three variables in the struct are a variable that holds the row for the block's position, a variable that holds the columns for the first edge of the block's position and the length of the block.

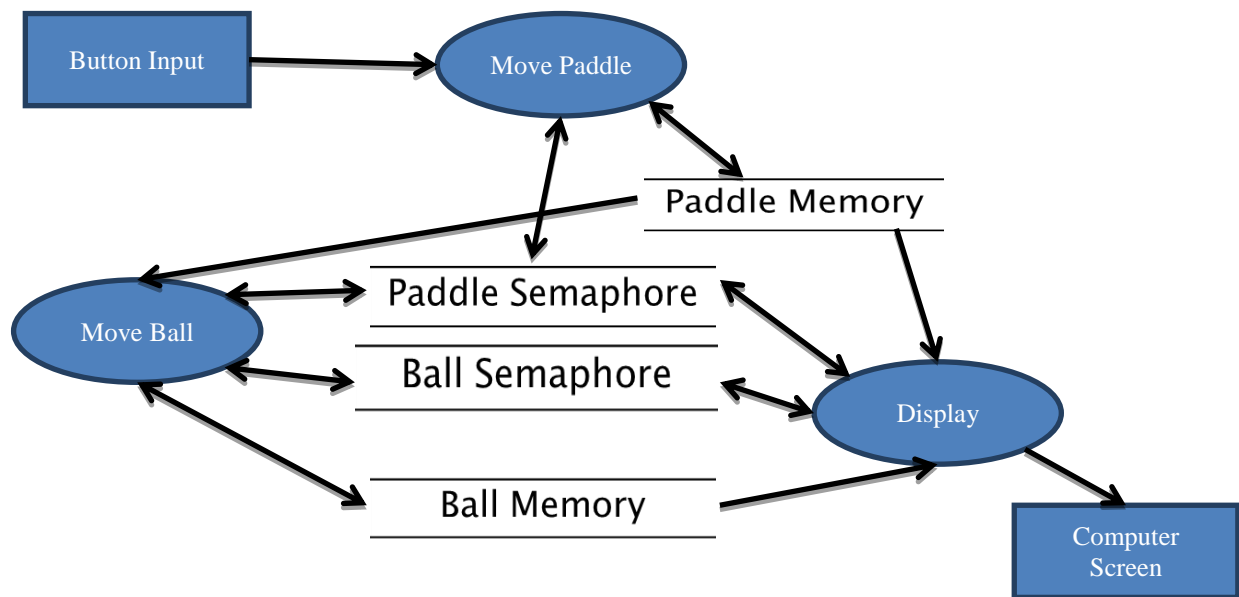


Figure 1 Diagram of Program's Structure

When the program starts, the program memory maps the GPIO registers of the board, sets the direction register for Port B so that the buttons on the auxiliary I/O board are inputs, initializes the semaphores for the ball and the paddle, and initializes all the variables of the targets such as the positions, colors, and the variable stating that none of the blocks have been hit. The program then creates the other threads and initializes the real time tasks. The program then executes in the structure shown in Figure 1. The Move Paddle task works by incrementing the variables that keep track of the last time that the paddle was moved. The task then polls the buttons and checks if one of the two movement buttons is pressed down. If one of the buttons is pressed, then the task checks to see if the paddle is already at the edge of the playing area and if

not, then the task waits for the paddle semaphore and increments or decrements the paddle location depending on whether the paddle should move left or right. The task then sets the variable that tracks the last time the paddle has moved in that direction to zero and sets the variable for the opposite direction to a number representing that the paddle hasn't moved in that direction in a long time. This means that if the paddle moves to the left, then the paddleLeft variable is set to 0 and the paddleRight variable is set to a large value. The task then posts the paddle semaphore and then waits for the period of the real time task. This task also checks if the button that signals that the game should stop has been pressed. If the button is pressed then the task exits this loop and causes the program to be ended.

The Move Ball task controls all of the movement of the ball. The ball struct has variables that correspond to the velocity of the ball. The Move Ball task has a variable that keeps track of the number of times the task has run without the ball moving. Once the counting variable is equal to the velocity value, then the ball moves in one step in the correct direction and then the counting variable is cleared. This means that if the velocity value is small then the ball will move faster, if the value is large then the ball will move slowly in that direction, and if the value is zero then the ball will not move at all in that direction. This method was used for moving the ball was used instead of simply changing the period of the task according to the velocity so that the ball could have different velocities in the horizontal and vertical directions but the same task could be used to move the ball in both directions. The task works by first checking if the vertical velocity is negative, meaning that the ball is moving down. If this is the case, then the counting variable for vertical movement is decremented. If the ball should then be moved, which means that the vertical counting variable is equal to the vertical velocity variable, then the task waits for the ball semaphore and the paddle semaphore. If the ball is directly above the paddle, then the task sets a flag that says that the ball has hit something and posts the paddle semaphore. The task checks if the paddle move variables are below a certain threshold which means the paddle has been moving recently and so then the horizontal velocity of the ball is changed depending on how the paddle has been moving and how the ball is moving. If the ball was not directly above the paddle then the task posts the paddle semaphore and then loops through all of the blocks to see if one of the targets is being hit. If the block is hit, then the variable stating that the block has been hit is cleared, the score is incremented, and the flag saying the ball has hit something is set. Once the task has checked if the ball has hit anything, the task checks if the ball has hit something and if so then the row variable of the ball is decremented so the ball moves upward and if not then the row is incremented so that the ball moves down. After the movement then the ball semaphore is posted. The task does the same thing for each of the other possible different directions that the ball could move: up, right, and left. Each of the different directions has different extra things that ball could bounce off of along with the targets. If the ball is moving up, then the task checks if the ball is bouncing off of the top of gameplay area. If the ball is moving horizontally then the task checks if the ball is bouncing off of the sides of the gameplay area. If the flag that says the ball has bounced vertically is set, then the task waits for the ball semaphore, multiplies the

vertical velocity by -1 so that the ball has the same velocity in the opposite direction, and then posts the ball semaphore. The same thing is then done for a flag that checks if the ball has bounced horizontally.

The task then checks if the ball is at the bottom row of the gameplay area. If the ball is then the variable for the number of lives the player has is decremented and the ball and paddle locations are initialized again at the middle of the play area. After checking if the ball is at the bottom, the task checks if the player's score is a multiple of 15, meaning all of the blocks have been cleared. If this is the case, the ball and paddle locations are restarted, the vertical velocity is divided by 2 while rounding up so that the ball travels about twice as fast, and the blocks are reset so that they haven't been hit yet. The task finishes by checking to see if the player doesn't have any lives left in the game, and if so then the whole game is re-initialized so that the score and speed of the ball is returned to the starting values. The task then waits for the next period.

The Display task works by using printf statements and ANSI escape sequences in order to clear the screen, set the cursor positions, and set the colors. The task starts by clearing the entire screen and then drawing the boundaries. The task then loops through the array for the targets and if the target hasn't been hit then uses the color variable inside the block struct and the location variables in the struct position the cursor. The task then loops for the size of the block and prints space characters in order to create the block. The task then displays the lives that the player still has at the top of the screen. The task then waits for the ball semaphore and then displays the ball and posts the ball semaphore. The task waits for the paddle semaphore, displays the paddle, and then posts the paddle semaphore. Finally the display task prints the score to the screen and then waits for the next period. Whenever a block or a line was needed to be displayed, the background color was set to the color the block was supposed to be drawn as and then spaces were printed. The ball was created by printing a capital "O" with a white text color and the black background. An example of the display is shown in Figure 2.

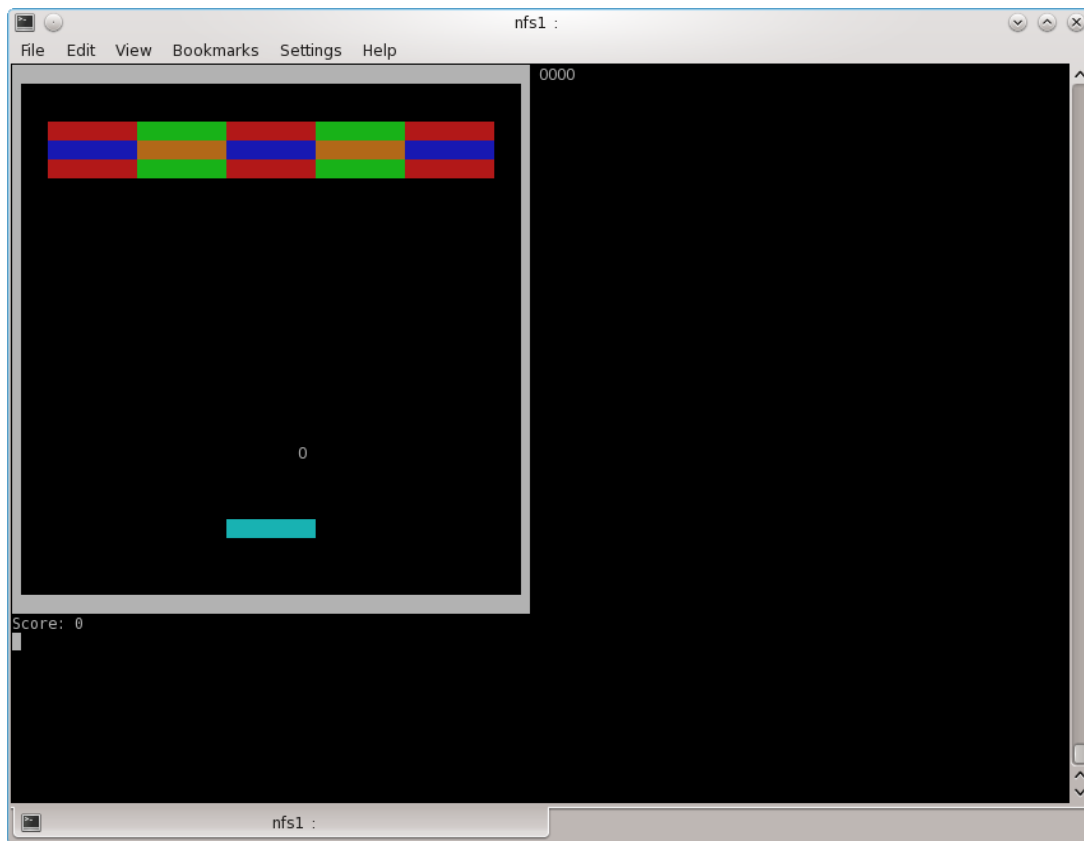


Figure 2 Example Gameplay

Real time tasks were used to for each of the different parts of the program so that the tasks could run at different period lengths so that the game could run correctly. If real time wasn't used for the move paddle task, then the paddle could move slower or quicker randomly depending on how quickly the computer was running. The move ball task had to be real time so that the ball would move with a constant velocity instead of speeding up and slowing down. The display task had to be real time so that the display updated fast enough and regular enough that large jumps weren't made by the ball or paddle thus making the game seem like it was lagging when it actually wasn't. The display task also has a lower priority than the other two tasks so that the display task is completed after the two movement tasks if they are all ready at the same time so that the display shows the last positions of the ball and paddle and the ball and paddle are shown in the correct position. The semaphores for the ball and paddle were put in place and were used whenever the two memory locations were used in order to make sure that one of the tasks didn't change the value while another task was checking what the value was. This makes sure that move ball task doesn't try to check if the ball is bouncing off of the paddle while the move paddle task is moving the paddle. The semaphores also make sure that the display task is showing the correct locations for the ball and paddle and that the locations of these objects don't change while the display is working which would mean that the ball or paddle was displayed in the wrong location.

Tests:

This program was tested by playing the game numerous times and checking to see if any bugs were encountered during the game. During the games, different scenarios would be tested to make sure that the game would respond accordingly e.g. trying to move the paddle when the ball hit the paddle and making sure the horizontal velocity changed in the correct way for all of the different possibilities and purposefully letting the ball fall to the bottom of the screen to make sure the lives were removed and that the game would restart when the player lost all of their lives. For any situations that could not be recreated while playing the game such as having a ball hit a block with a specific velocity, the initialization part of the program was changed so that the ball would start in that situation and then the game would be tested to see if the game acted as it was supposed to. In order to make sure that the game would act correctly when all of the blocks had been cleared, the paddle length was changed so that the paddle would cover the entire playing area so that the ball would always hit the paddle and all of the blocks would be hit to make sure that the blocks would be restarted and the ball's velocity would be increased. The game passed all of the tests and no problems occurred while test playing the game.

Conclusion:

The game of Breakout was successfully implemented on the TS-7250. No problems were seen in the many tests that were conducted on the game. One thing that could possibly be changed is the way in which the movement of the ball was done. It could possibly be implemented by using two real time tasks, one which controlled vertical movement and one which controlled horizontal movement, that would be run with different frequencies depending on how fast the ball was supposed to move. This wasn't done because there could be a problem when the ball was moving diagonally and was on the edge of block and the ball would then seem to move first in one horizontally and then vertically thereby moving around the block instead of moving diagonally and hitting the block. If this problem with movement in different directions at the same time, then the code could possibly be more efficient by running the move tasks whenever the ball actually had to move. Another change that could be easily implemented would be to create different arrangements of the target blocks whenever the all of the blocks were cleared, this would just involve changing the location variables for each of the blocks whenever the blocks were reset. The game works correctly with the finished project with some possible improvements that could be made later.