COMP206 Computer Architecture

Atari Breakout Game on MIPS
Project Report

Tolga Neslioğlu	042301100
Cem Alp Özer	042301097
Birhat Tutuş	042301158
Sudenur Bilgin	042101060

MEF University, Istanbul, Turkey

Explanation & Features

The project is an implementation of Atari Breakout, a one-player arcade game where the goal is to destroy a wall of bricks by bouncing a ball off a paddle. The player controls a horizontal paddle at the bottom of the screen to keep the ball in play. If the ball passes the paddle and hits the bottom edge, the game is over.

Gameplay and Mechanics

In this project, the Breakout gameplay was recreated on a 32×32-pixel bitmap display. The game starts with a ball positioned above the paddle and a formation of bricks at the top of the screen. The player uses keyboard controls to move the paddle left and right, bouncing the ball upward into the bricks. Each time the ball hits a brick, the brick is "broken" (erased from the screen), and the ball bounces away in the opposite vertical direction. The ball also bounces off the side walls and the top border of the playfield, using simple physics (inversion of the respective X or Y velocity component upon collision).

- The ball recognizes and responds to collisions with the paddle, walls, and bricks. Upon hitting a brick, the brick is removed, and the ball's direction reverses vertically. Hitting a side wall reverses the horizontal direction, and hitting the top border reverses the vertical direction downward.
- The paddle collision simply reflects the ball upward. (We discussed adding more complex bounce physics where the bounce angle varies based on where the ball hits the paddle. But in the final implementation the bounce off the paddle is a standard vertical reflection.)

Game Features

In addition to the basic paddle-and-ball mechanics, there are several features in the game that are inherent from the skeleton code:

- **Paddle Control:** The player can move the paddle using keyboard keys. Player can press the “A” key to move the paddle left and “D” key to move it right. The paddle’s movement is bounded within the screen (it cannot go beyond the left or right walls).
- **Brick Layout:** The playfield includes a structured arrangement of bricks. There are seven rows of bricks at the top of the screen (just below a top border area). All bricks are identical in size and color in the game.
- **Pause and Quit:** In the game there is simple pause and quit controls for convenience. Pressing “P” at any time will pause the game (freezing the ball’s motion) and can be pressed again to resume play. Pressing “Q” will immediately end the game (triggering an instant game over.

Additional Features Implemented

- **Score Tracking:** A scoring mechanism is implemented. Every time the player’s ball hits and breaks a brick, the score counter increases by 1. The current score is continuously tracked in a variable. At the end of the game the score can be viewed in the console output of the simulator (Program prints “SCORE: X” to the console, where *X* is the current number of bricks destroyed). The score starts at 0 at the beginning of each game and resets to 0 upon restarting. (Our score display currently supports single-digit scores 0–9.)
- **Game Over Screen:** The Game Over screen, which appears when the game ends, was implemented. A 32x32 pixel "Game Over" graphic is displayed on the simulator's screen. A bitmap image spelling out "Game Over" was created and integrated into the

assembly program as data. When the player loses, this image is drawn to the screen, providing a clear visual indication of the game's end state. In our version, the Game Over screen also includes modified text alongside the graphic, and the final score is output to the console.

- **Restart Functionality:** After the Game Over screen is shown, the player has the option to restart the game without needing to reset the simulator or reassemble the code. A restart function that resets all game state variables to their initial values when the player presses the "R" key was implemented. This includes clearing the previous score, resetting the ball and paddle to their starting positions, restoring the initial ball speed, and redrawing the initial scene (bricks, paddle, ball) with `restart_game` routine. Once the player presses R after a game is over, a new game begins just like the original start.
- **Dynamic Speed Increase:** To enhance the game's difficulty progression, a mechanic where the game speed increases after the player scores 5 points was added. In practice, this means the ball movement accelerates once the player has broken 5 bricks. Technically, the game's loop delay (the interval controlling frame updates) is shortened at that milestone. The implementation checks the score each time a brick is hit; when the score reaches 5, a routine is invoked to adjust the speed variable. Specifically, the delay is reduced between updates by 25%, making the ball travel faster. This one-time speed boost raises the challenge for the player as they progress. The mechanism was implemented with consideration of assembly arithmetic, since floating-point operations are not readily available. We achieved the 0.75× delay adjustment by using integer math (multiplying by 3 and dividing by 4) to scale the speed value.
- **Bug Fix – Paddle Movement Freeze:** During development, we encountered a major bug inherited from the base code. If the player moved the paddle at certain moments, the ball would freeze in place (ceasing to move) until player stops moving the paddle. Upon investigating the original assembly logic, it is found that the collision handling

and paddle movement input interfered with each other. In the base code, the ball's direction might not have been updated correctly if the paddle's position changed in the same frame, resulting in a velocity of zero and a stuck ball. The bug was fixed by adjusting the order of operations and conditions in the game loop. The corrected code makes sures that when the ball hits the paddle, its vertical direction is inverted (bounces up) regardless of simultaneous paddle movement, and the velocity variables are never set to an invalid zero state. Safeguards were also added so that the ball's position is always updated each cycle, even if multiple inputs occur at once.

Implementation Details

- **Game state initialization**

  Initial game state (paddle positions, ball position/velocity, brick layout, score, and speed) is set up by the `initialize_game` routine at program start.
- **Input polling**

  Player keystrokes (paddle controls and restart command) are read by the `read_input` routine each frame.
- **Paddle updates**

  Paddle positions are adjusted by the `update_paddles` routine based on the most recently polled input.
- **Ball physics**

  Ball position and velocity are computed by the `update_ball` routine every cycle, applying basic vector movement.
- **Collision handling**

  Collisions with walls, paddles, and bricks are detected by the `check_collisions` routine. Brick removal and score incrementation are triggered within this routine.
- **Speed adjustment**

  The game's frame-delay value is modified by the `adjust_speed` routine once the score threshold (5 points) is reached, producing a permanent speed increase.

- **Scene rendering**

  The playfield—including borders, bricks, paddles, and the ball—is redrawn each frame by the `draw_scene` routine to the 256×256 bitmap.
- **Game-over display**

  When a miss occurs, the `display_game_over` routine is invoked to render the custom "Game Over" graphic and output the final score to the console.
- **Restart logic**

  Upon detection of the restart key, the `restart_game` routine is called to reset all dynamic variables (score, positions, speed) and jump back into the main game loop, enabling a fresh session.

Key Variables and Object Properties

- **screen_base**

  The base address of the 256×256 bitmap display is held in `screen_base` for all pixel-drawing routines.

- **brick_map**

  The layout of bricks is stored in the `brick_map` array (one word per brick), with each entry indicating whether the brick is present (and, if applicable, how many hits remain).

- **paddle1_x**

  Paddle's horizontal position is recorded in `paddle1_x`.

- **paddle_width**

  The constant width of both paddles is defined in `paddle_width` to simplify collision checks and drawing.

- **ball_x, ball_y**

  The ball's current coordinates are tracked by the words `ball_x` (horizontal) and `ball_y` (vertical).

- **ball_dir_x, ball_dir_y**

  The ball's movement direction (±1 in each axis) is held in `ball_dir_x` and `ball_dir_y` and is inverted upon collisions.

- **score**

  The player's score is accumulated in the `score` variable and is incremented by one each time a brick is destroyed.

- **frame_delay**

  The delay between game-loop frames is stored in `frame_delay`; once the score reaches five, this value is scaled down to speed up gameplay.

- **restart_key**

  The last keycode read from input is saved in `restart_key` so that an 'R' press can be detected to trigger the restart routine.

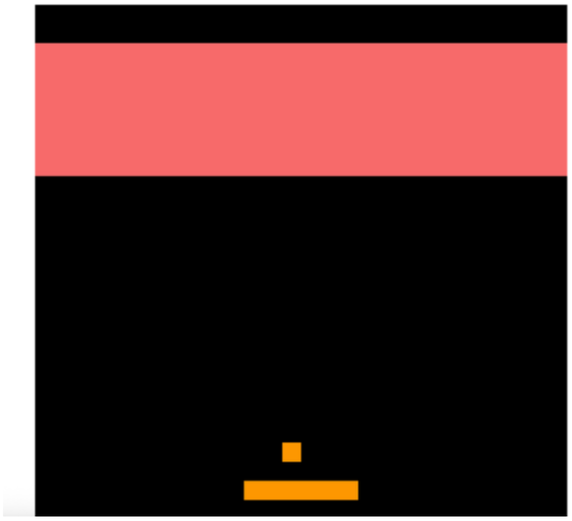
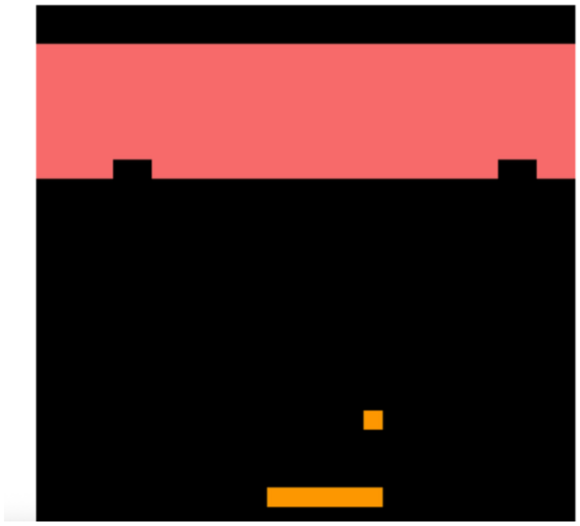
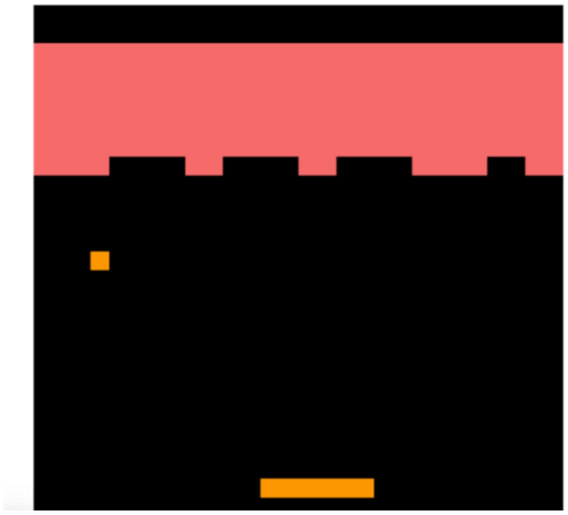
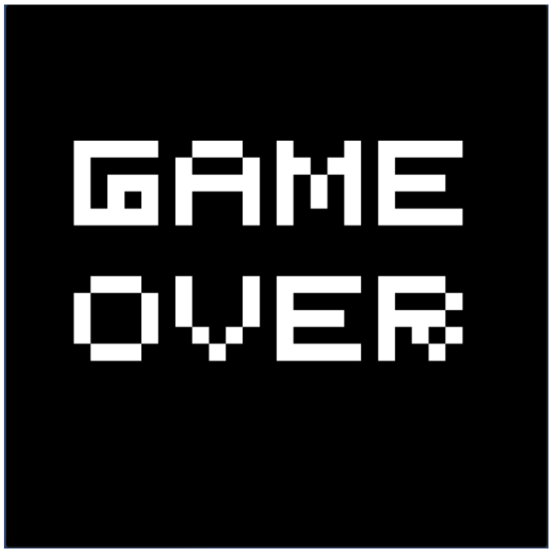
- **game_over_flag**

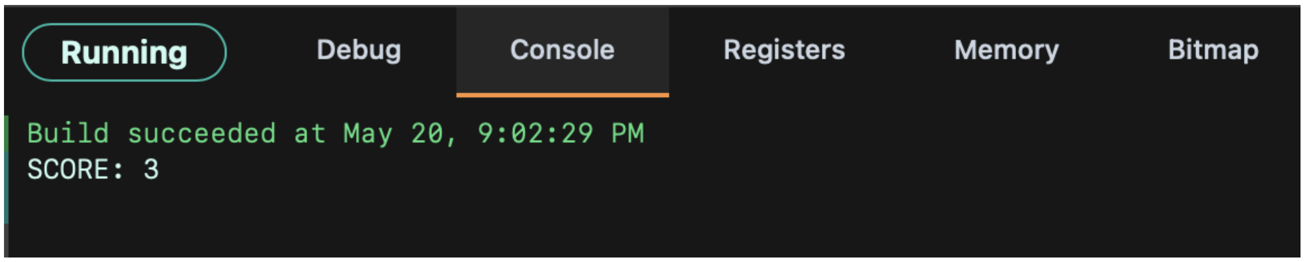
  A boolean word (0 or 1) is set in `game_over_flag` when a miss occurs, causing the Game Over screen routine to be invoked.

Object Properties

- **Paddle** object is defined by:
  - *Position* (`paddle_x`)
  - *Width* (`paddle_width`)
- **Ball** object is defined by:
  - *Position* (`ball_x`, `ball_y`)
  - *Velocity/Direction* (`ball_dir_x`, `ball_dir_y`)
  - *Speed control* (`frame_delay`)

In-game Screenshots



```
Build succeeded at May 20, 9:02:29 PM
SCORE: 3
```

4-Resources Used

[1] ramyzhang, “GitHub - ramyzhang/mips-breakout: A cutesy clone of Atari Breakout written in MIPS assembly!,” *GitHub*, 2023. https://github.com/ramyzhang/mips-breakout.

Meeting Plans

April 23	30 mins	Deciding on project topic & literature review.
April 30	30 mins	Finalizing project topic.
May 7	1 hour	Literature review, base (skeleton) code review, discussing possible features to add.
May 15	1 hour	Implementing selected features into base code and labor authentication. - Ball freeze bugfix (Tolga) - Game over screen (Tolga) - Score system (Sude) - Restart feature (Birhat) - Dynamic game speed increase (Cem)
May 19	1 hour	Discussing the report and recording the demo.

Demo

Video Link: https://youtu.be/-y0wPLZladM