The Breakout Game

By
Umang Deshpande and Akshay Hegde

June 2017

1 Objectives

- Understanding RTOS and Statechart Principles.
- Design and Development of the Breakout game.
- Implementation of the game on the console.

2 Prerequisites

- Basics of RTOS, familiarization with tasks, semaphores and task scheduling.
- Drawing up of a statechart from the game design statement (Please refer to the Bomb Timer example in the Resources Section).
- Implementation of visual statechart in code using switch case construct (Please refer to the link given here).
- Creation of GLCD graphic for game objects(Please refer to the Mikroelektronika GLCD Font Creator Example in the Resources Section).

3 Game Design Statement

The overall objective is to design the classic Breakout Game for the Tiva C Game Console, using the onboard peripherals. The breakout game has a rectangular paddle at the bottom of the screen off which a moving ball ricochets off. At the top of the screen, it consists of rows of bricks. The victory condition is to clear the row of bricks, by hitting it with the ball. Our implementation of the game is to have 5 types of bricks:

- *HARD* brick - Takes 3 hits to disappear.

- *MEDIUM* brick - Takes 2 hits to disappear.
- *EASY* brick - Takes 1 hit to disappear.
- *MAGIC1* brick - Delivers a hit to all surrounding bricks.
- *MAGIC2* brick - Increases paddle size for 10 seconds.

There are 3 difficulty levels:

- *EASY* - 3 lives, higher proportion of EASY bricks.
- *MEDIUM* - 2 lives, higher proportion of MEDIUM bricks.
- *HARD* - 1 life, higher proportion of HARD bricks.

Implicit above is an implementation of a life system(allowed number of retries for the player). There are three speeds for the ball:

- *SLOW* - Ball moves slowly.
- *MEDIUM* - Ball moves at an average speed.
- *FAST* - Ball moves very fast.

Additionally, the game has the following screens:

- Entry Cutscene
- Menu Screen
- Instructions Screen(Accessible from the menu)
- Settings Screen(Accessible from the menu)
- Gameplay Screen
- Victory Screen
- Game Over Screen

4 Relevant Theory

4.1 Statechart Solution

Since the game has several active components at the same time, it runs several parallel state machines each of which may or may not be independent of the other. The different state machines are as follows:

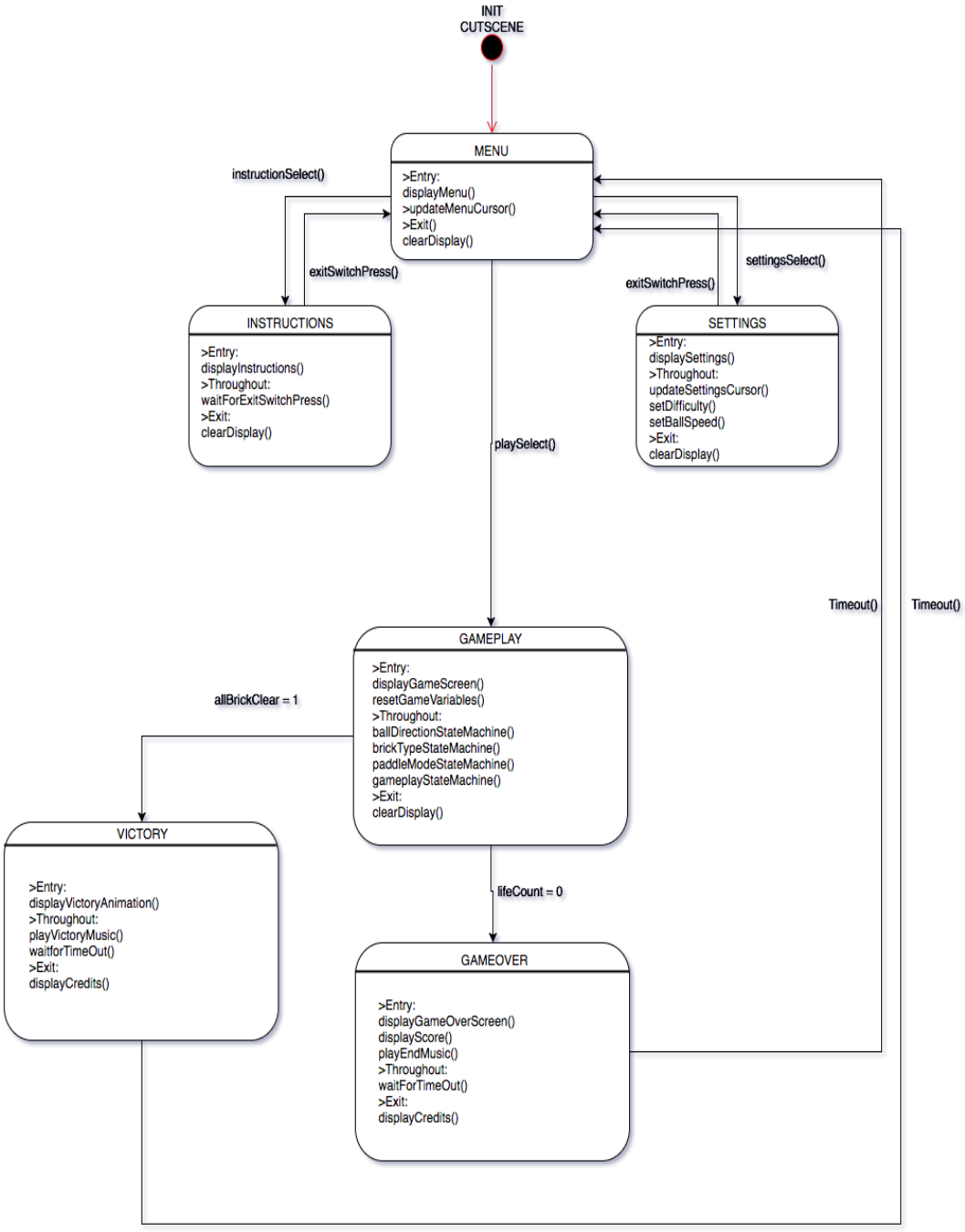

Fig (a): Game Screens State Machine

The overall states in which the game exists is as shown in Fig (a). A brief explanation of each state is as follows:

- **MENU** State:
  This acts as the GUI for the user, and follows the initial cutscene(The Cutscene displays the cutscene graphic while playing Music). The Menu Screen is then displayed, and switch presses are used to move the cursor.
  RIGHT Switch press is used for selection.
  UP Switch press is to move cursor up.
  DOWN Switch press is to move cursor down.
  A screenshot of a sample *MENU* state is as shown

Fig (b): Menu Screen

- **INSTRUCTIONS** State:
  This displays instructions of gameplay for the user. *LEFT* Switch Press = Back.
  A sample *INSTRUCTIONS* screen:

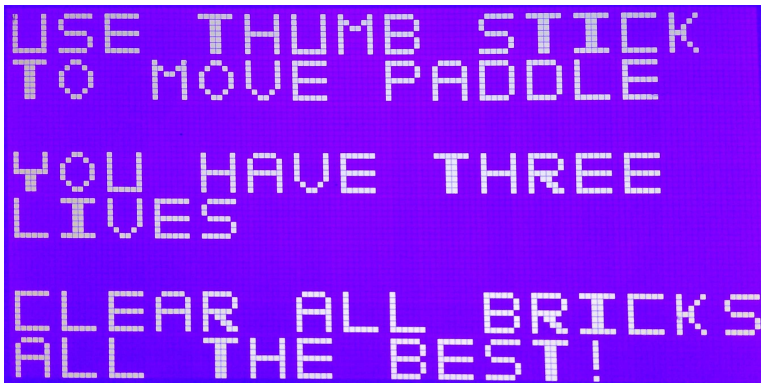

Fig (c): Instructions Screen

- **SETTINGS** State:
  This allows for change of difficulty and ball speed settings for the player.
  UP Switch moves cursor up, and at topmost position, acts as back Switch.

DOWN Switch moves cursor down, and at bottommost position, acts as back Switch.
LEFT selects easiest setting, RIGHT selects toughest setting, HAT selects medium setting.

Fig (d): Settings Screen

- **GAMEPLAY** State:
  This is the actual gameplay screen running internal state machines.
  Thumbstick moves paddle left or right.
  The below screenshot shows a paddle at the bottom, a ball in transit in the *UP LEFT* direction and certain bricks are removed.

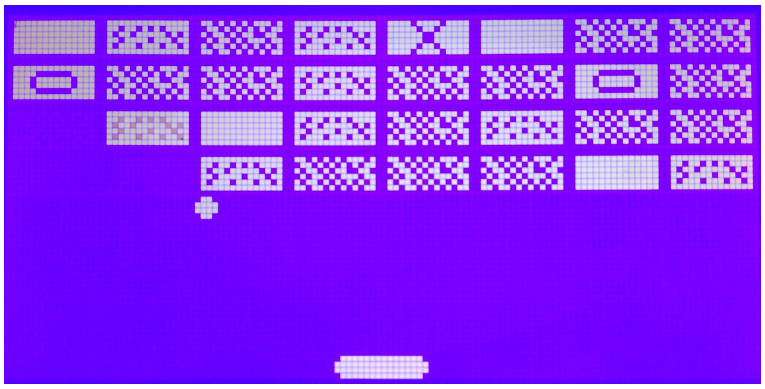

Fig (e): Gameplay Screen

- **GAME OVER** State:
  Once the player runs out of lives, the game over screen is displayed which accepts no input, but plays certain short music. Then returns to *MENU* State. It can be as shown:

Fig (f): Game Over Screen

- **VICTORY** State:
  Once the player clears all bricks, a victory music is played over a victory screen, following end of music, game returns to *MENU* State.

Fig (g): Victory Screen

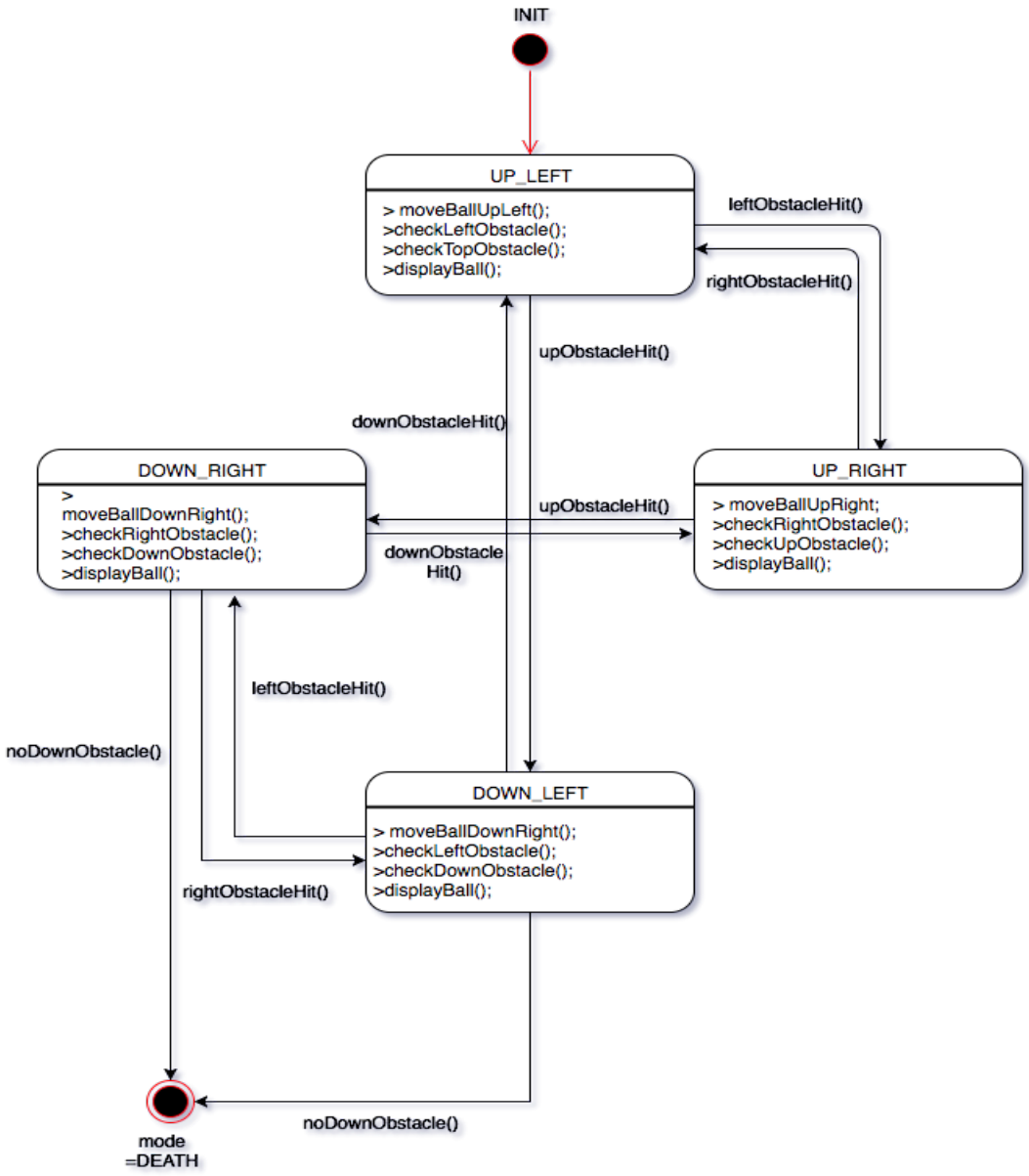

Fig (h): Ball Direction State Machine

Fig (h) shows the various modes of motion of the ball. Unless an obstacle is detected, the ball continues to be in a particular state/ direction, as shown. If the obstacle is a brick, *hit* variable is decremented. If an obstacle is not present at the bottom, a life is lost. If an obstacle is present at the bottom, it needs to be the paddle which is tracked with the ADC thumbstick input.

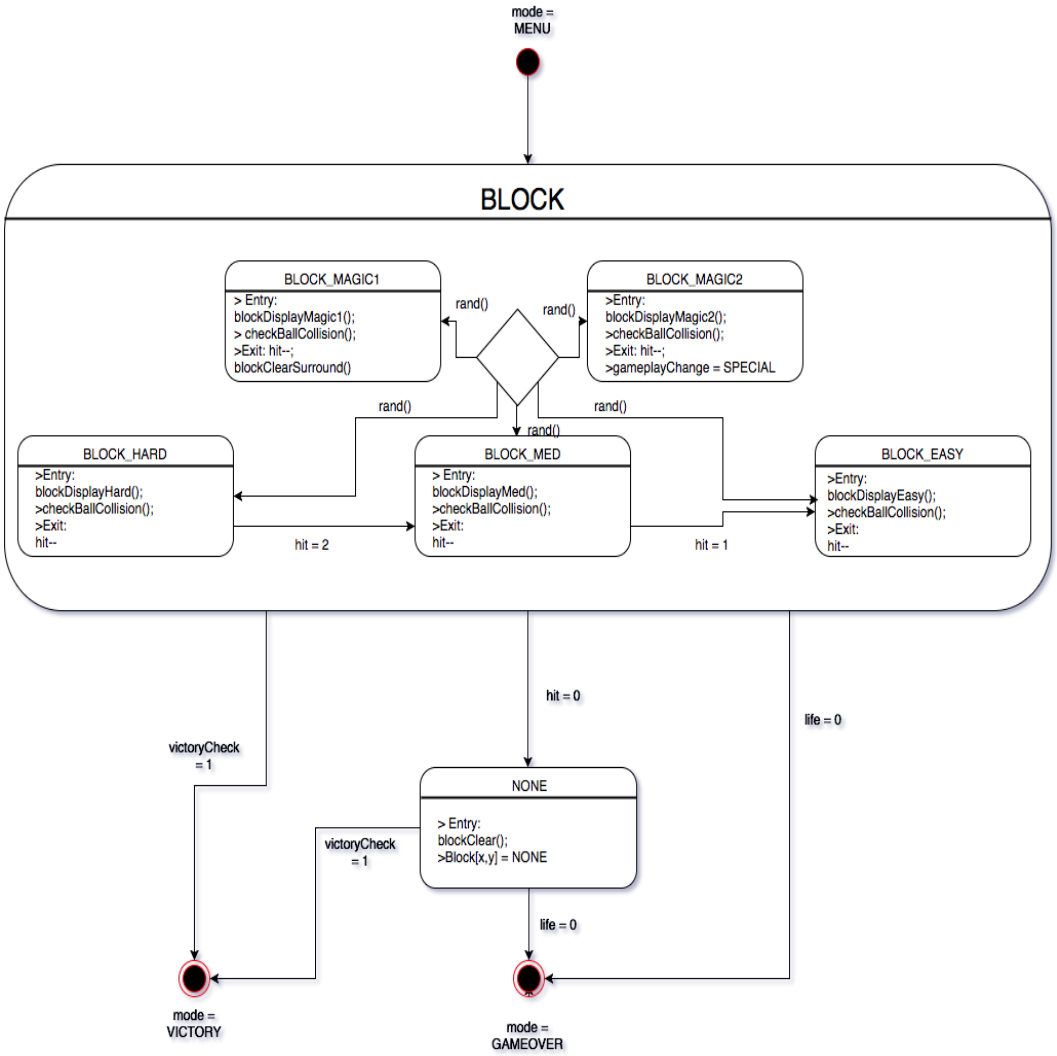

Fig (i): Brick Type State Machine

The above state machine denotes switching between various states of each block. Initially, a randomizer randomizes the entire block wall. Thereafter, based on ball hit, different blocks are eliminated. The brick state machine is terminated upon *DEATH* and *VICTORY*.

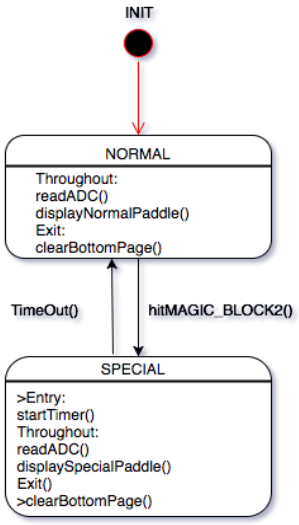

Fig (j): Paddle Type State Machine

The visualization above represents the two different states for the paddle, the special extended state for 10 seconds once the ball hits BLOCK_MAGIC2 and the normal state otherwise. Each state reads from ADC and displays the paddle at the appropriate position.

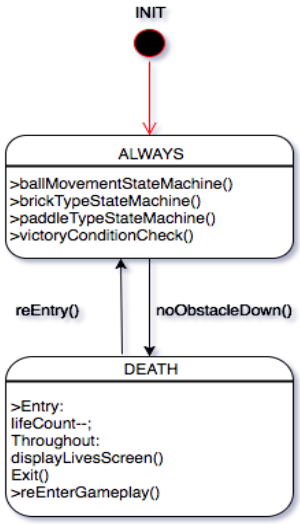

Fig (k): Gameplay Internal State Machine

The above state machine shows two different states within gameplay. Gameplay continues till the player dies, upon which, certain tasks are performed.

4.2 RTOS Implementation

The Breakout Game can be implemented using two tasks: *readInput()* and *displayOutput()*

4.2.1 readInput() Task

Handles all input switch press detection and ADC thumbstick movement. The behaviour is different in different game screens, hence consists of an internal state machine which defines behaviour at different states, implemented using switch case.For example, *SWITCH* press is an event in the *MENU* state but not in the *GAMEPLAY* state.

4.2.2 displayOutput() Task

Handles output screens to be displayed on GLCD corresponding to different states. Also handles LED outputs, Buzzer beeps and music, and also the vibration motor output. Hence different screens have different outputs, it consists of an internal state machine which determines the menu screen to be displayed in *MENU* state, but Game over screen to be displayed, alongwith appropriate music in *GAME OVER* state.

5 Procedure

1. Include all the relevant header files in your code. Ensure that the following header files are present:

```
#include "Console/consoleInit.h"
#include "Console/glcd.h"
#include "Console/gameDisplay.h"
#include "Console/tones.h"
#include "Objects/gameObjects.h"
#include "Screens/gameScreens.h"
```

   The above libraries contain direct-to-GLCD font libraries, pre-made music and beeps, game graphic objects and screens in the form of accessible functions. The libraries can be downloaded from here.

2. All necessary initialization, including Timers, Interrupts, Clock, ADC, GPIO, UART, etc. can be performed using function _init_() from consoleInit.h. Edit consoleInit.h to initialize other peripherals in addition to the ones provided.

3. Initialize each of the various states using enumeration as in the following example:

```
enum modes
{
    /* Different game states(Classified because of
    different I/O behaviour in each state) */
    MENU,
    INSTRUCTIONS,
    SETTINGS,
    GAMEPLAY,
    VICTORY,
    GAMEOVER
};
// Initialization
enum modes mode = MENU;
```

4. Declare two tasks(empty), with corresponding semaphores *readInput()* and *displayOutput()*.
5. Use Timer 2A Interrupt for task scheduling and post semaphores at appropriate instances in time.
6. Now, in each task, implement the overall game state machine, with its various internal state machines, as in the statecharts given in Section 4(Switch Case or State Table Implementation can be used).

6 Demo and Submission:

- Implement appropriate statechart within each task and use appropriate task scheduling in RTOS.
- Show working game demonstration to TA.

All The Best