LET THE BALL DROP

Created By: Thomas Davies - 200912136

An embedded system gaming platform developed for ELEC 2645.

Contents

1.0 Statement of Academic Integrity 2

2.0 Abstract 2

3.0 Introduction 2

3.1 Embedded Systems 2

3.2 Project Overview 3

4.0 Project Specification 3

4.1 Project Aims 3

4.2 Project Objectives 4

4.3 Project Constraints 4

4.4 Specification 4

4.5 System Diagram 5

5.0 Development 6

5.1 Hardware 6

5.2 Software 7

5.2.1 Overview 7

5.2.2 Function Documentation 10

6.0 Testing 17

6.1 Hardware Testing 17

6.2 Hardware Refinements 18

6.3 Software Testing 18

6.4 Software Refinements 18

7.0 Conclusion 19

8.0 References 21

9.0 Appendix 22

9.1.0 Code 22

9.1.1 Main.h 22

9.1.2 Main.cpp 25

9.1.3 GameScreen.h 36

9.1.4 GameScreen.cpp 42

9.2 Schematic 50

9.3 PCB Design 51

1.0 Statement of Academic Integrity

Plagiarism in University Assessments and the Presentation of Fraudulent or Fabricated Coursework

Plagiarism is defined as presenting someone else's work as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance. Fraudulent or fabricated coursework is defined as work, particularly reports of laboratory or practical work that is untrue and/or made up, submitted to satisfy the requirements of a University assessment, in whole or in part.

Declaration:

• I have read the University Regulations on Plagiarism [1] and state that the work covered by this declaration is my own and does not contain any unacknowledged work from other sources.

• I confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.

• I confirm that details of any mitigating circumstances or other matters which might have affected my performance and which I wish to bring to the attention of the examiners, have been submitted to the Student Support Office.

[1] Available on the School Student Intranet

2.0 Abstract

The basic aims of 'Let the Ball Drop' were to create a small, portable, battery efficient gaming platform that was both fun to play and extremely addictive. The embedded system is based on the popular mBed LPC1768 powered from a 9V battery after regulation through a buck converter. Input and output from the system are provided using a joystick with push button and Nokia 5110 display respectively. The hardware and software of the embedded system were designed from ground up using professional methodologies and practices. Overall, the system was a great success resulting in a design that met all set aims and specifications.

3.0 Introduction

3.1 Embedded Systems

An embedded system is a combination of hardware and software that is specifically designed for a particular objective. Embedded systems can be found in all devices that have some form of digital display such as a watch, DVD player or microwave. The embedded system is commonly housed on a single chip for space saving and durability [1].

The main difference between an embedded system and personal computer is that the latter is designed with versatility and change as a feature. A PC user may be able to listen to music, while browsing the internet, and downloading a video simultaneously; whereas an embedded system is designed to perform a single (or few) dedicated task(s), and cannot be changed once deployed. This difference makes embedded systems ideal for low-cost and power applications, where a versatile processor is unnecessary.

At the heart of an embedded system is a microcontroller which contains all the key components of a computer but in a single package. The main components of a PC are the processor, main memory (RAM), secondary memory, clock, and peripherals, all of which are integrated into a single package for a microcontroller [4]. Microcontroller are designed to be small, low-power and inexpensive alternatives to the traditional computer.

3.2 Project Overview

"Let The Ball Drop" is a small hand-held, battery powered, embedded gaming platform. The system allows a user to play a fun, and addictive game, controlled using a joystick and displayed using a Nokia LCD screen. The game is a simple one, where the user controls a ball and must avoid being pulled to the top by the rising platforms. Each platform has a small gap in it for the ball to fall through, the user must navigate to these gaps. As the game progresses the speed of these platforms gradually increases, making it more and more difficult to reach the gaps in time. For replay value and difficulty the location of gaps in each platform are randomly generated. Beyond the basic mechanics of the game, the system also features a pause menu and high score management, where the user can enter their initials and save their scores.

The problem with gaming on a phone is that it drains battery quickly and can lead to a dead phone when it is actually needed. By using an embedded system specifically developed for gaming, users can rest assured that their phone will have the battery it needs to make a call while still being able to have fun playing games. "Let The Ball Drop" was designed to be small, portable device with a long lasting battery life, the system is also not reliant on AC power to charge, instead boasting the ability to hot swap a dead 9V battery for a new one in seconds. The near-instant boot up speed (relative to phones) and portable design allows a future where "Let the Ball Drop" could be played on every bus, plane, or train around the world.

Embedded systems are generally dedicated to a single task, and optimized for only that task. "Let The Ball Drop" is an examples of such a system as from the design phase of the PCB to the final touches on software, gameplay has been at the core of design. The system features all the key components of an embedded system, microcontroller (mBed), power regulation, input (joystick), and output (screen and buzzer).

4.0 Project Specification

4.1 Project Aims

The project aims are the first stage in design of the embedded system, the point below are broad statements to explain the general intentions of the project. These points are unaltered from the original ideation phase.

- To develop a small portable gaming system
- To create a fun and easily played game with a certain "addiction" aspect
- To display game graphics in a fluent and intuitive manner
- To maximize battery life by using as little power as possible
- To layout the PCB in a way that optimizes usability.

4.2 Project Objectives

The second phase of idea development was to take the project aims and refine them into objectives. These are the steps needed to take needed to reach each of the project aims.

- A Nokia LCD will be used to display graphics and text.
- To animate graphics regularly, various tickers will be used.
- To minimize power-usage
  - An efficient switching regulator will be used
  - Unused peripherals and clocks will be turned off
- To optimize usability the PCB layout must be designed from successful prototypes and testing
  - Screen will be oriented vertically
  - Joystick will be placed such that it does not hinder view of screen
- Text will be used to display instructions to user and score throughout play.
- Game will be based on popular “Fall Down Game” available on Android and iPhone
- To keep game interesting each replay will be “random”
  - RTC must be kept running for proper “random” number generation

4.3 Project Constraints

In an ideal world a project is limited only by a designer’s imagination, but in reality there are various constraints due to cost, time, and availability. For this project the project constraints were clearly defined by Dr.Craig A. Evans, and are shown below.

- Limited to using a Joystick and Pushbuttons for user input
- Can use Nokia 5110 LCD display
- Project must be powered by a PCB-mounted 9V Battery (PP3)
- Must use a 9-to-5 V Buck converter (MC34062 and associated components)
- Can use 3 V coin cell as keep RTC backup
- Two-layer PCB board is limited to a size of 100mm by 80mm with track spacing of 10 mil and polygon isolation of 0.01 mil.
- Have access to:
  - 3 mm LEDS (red, green, blue)
  - 1 k-Ohm resistors
  - Push buttons
  - Slide switch (SPDT)
  - Piezo buzzer
  - 10 K Potentiometer

4.4 Specification

With project constraints developed and a list of objectives that must be met, the specifications can be developed. The specifications view each objective, and using the project constraints, develop a method to meet these objectives.

- The device will be powered by a single 9 V battery
- A 9-to-5 V switching regulator will be used to step down and regulate battery supply
- A Nokia 5110 LCD will be used to display graphics and text

- The base library N5510 by Craig Evans will be extended and overridden by a custom class to allow further functionality to specific game and vertical orientation of LCD.
- An array will be used to track location of each platform
- A start screen, instruction screen, then countdown will appear before gameplay
- User will be able to pause game by activating joystick button.
- High scores will be written to a csv and can be viewed by user
- A separate ticker for moving platforms, moving the ball, advancing the level and refreshing the screen will be used.
- Losing the game (inevitable) will activate a lose screen and audio feedback through buzzer
- Joystick readings will be taken in main loop with a timer attached to control speed of ball
- Ethernet and all other unused peripherals will be turned off to save power.

4.5 System Diagram

The system diagram in figure 1, shows a high level look at the overall circuit of the embedded system. Details on each of the hardware blocks can be found in section 5.1.

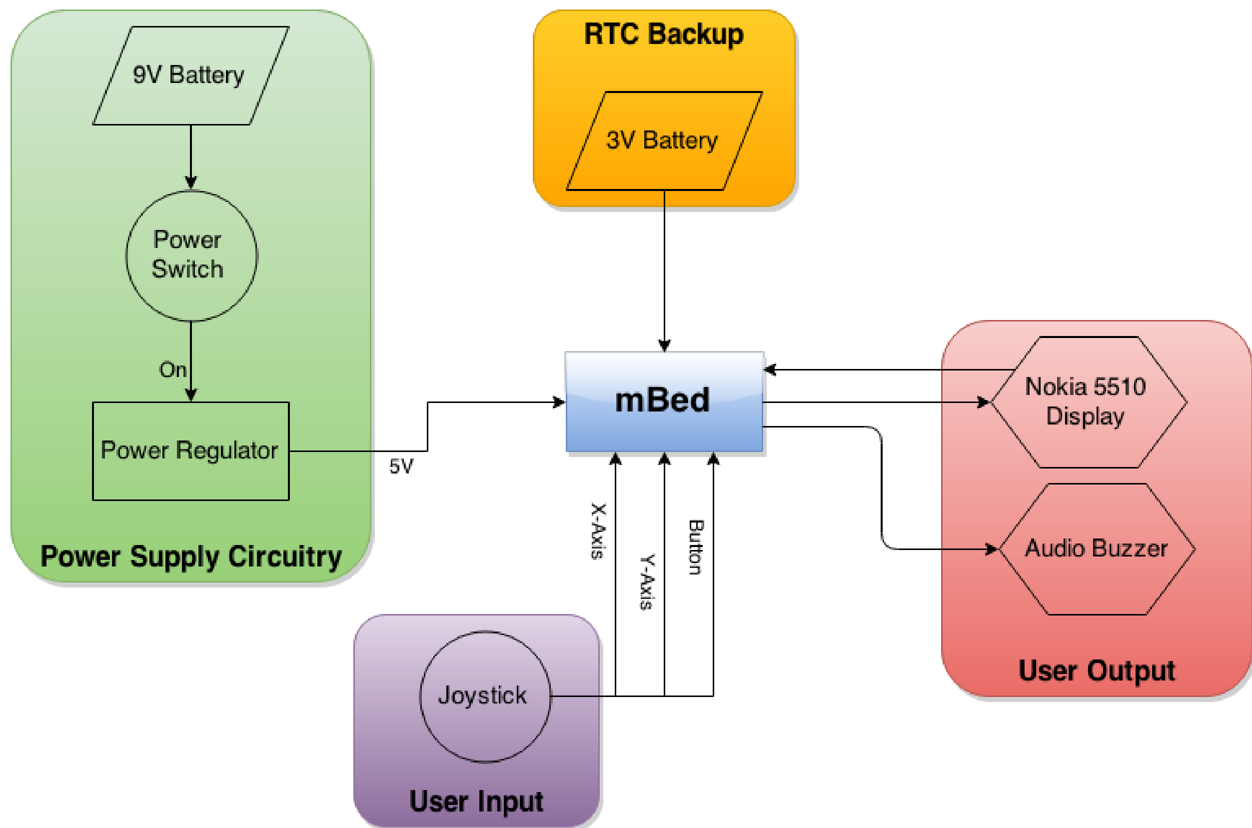


Figure 1: Overall system block diagram.

5.0 Development

5.1 Hardware

The block diagram in section 4.5 gives a high level overview of the key components within the embedded system. These blocks are power supply circuitry, user input, user output, RTC backup and the microcontroller. Alone each of these components serves a single purpose but when brought together, the system is capable of producing a fun and complete gaming experience.

The power supply circuitry has the task of regulating the supplied 9V battery to an mBed usable 5V DC signal. The signal is stepped down from 9V to 5V using a MC34062 based buck converter circuit. In theory the step-down should be to a flat 5V but in practice the regulator was measured to output 5.12V which is still within mBed specification and acceptable for use. The power regulation circuit schematic can be seen in figure 2.

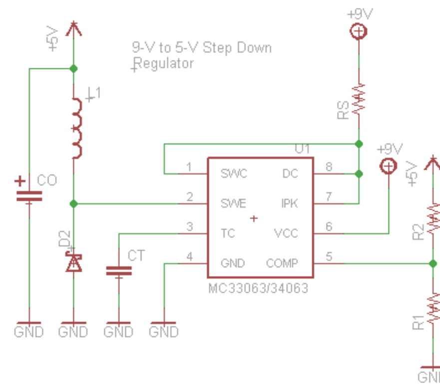


Figure 2: 9 to 5V buck converter circuit.

A switching regulator setup was chosen over a linear regulator due to the improved efficiency and smaller form factor. The PCB still featured a linear regulator back-up in case of failure, but after testing this was deemed unnecessary.

Although a switching regulator could be implemented using discrete components, it is much easier and reliable to use a switching regulator IC. In this case, the MC34063 was used, with supporting circuitry to produce desired topology. A breakdown of the components in the circuit along with values, how they were found, and usage can be seen in table 1.

Component	**Value**	**Usage**
Rs	0.5 Ω	Sense Resistor – sets peak current
Co	100 µF	For energy storage
L1	151 µH	For energy storage
Ct	47 µF	Timing Capacitor
R1	1 kΩ	Voltage Divider
R2	3 kΩ	Voltage Divider

Table 1: table of passive components with usage and value.

Each component value was calculated using the equations given in the MC34062 datasheet. The Schottky diode (D2) which is used as a path to ground when the 'switch' is closed. A key consideration when designing the power circuit on PCB was to keep lines between each component as short as possible. The length of lines introduces intrinsic inductance which can alter the performance of the regulator circuit.

Figure 3: Embedded system front view.

All user input to the embedded system is through the joystick potentiometers and click button. The joystick is used for game navigation, while the button is used for certain actions. There are three connections from the joystick to mBed pins. The two joystick

potentiometers are connected to pin 15 and 16, while the button is connected to pin 17.

User output is given through two devices, the Nokia 5110 screen and a buzzer. The Nokia screen is controlled through a synchronous serial interface similar to SPI [8]. There are clock (SCLK) and data (DN) input lines as well as an active-low chip select (SCE). Another input (D/C) informs the display whether the data received is a command or displayable data. The screen is powered via a digital pin, to allow on/off control. The screen backlight is also controllable from a PWM output. The buzzer will be controlled by a simple manual PWM signal from digital pin p5 [7].

The 3.3V watch battery is used to power the on-board real-time-clock which is used to keep track of time and date. The RTC is also used as a seed when generating random numbers, without a battery gameplay would be the same each time.

The final hardware component is the mBed LPC1768 which is the main processing unit of the embedded system. The mBed is based off a 32-bit ARM® Cortex™-M3 running at 96MHz. Also featuring 512KB FLASH, 32KB RAM and interfaces including USB host and device, SPI, I2C, PWM and others [2].

Figure 4: Embedded system back view.

The PCB layout of the embedded system was carried out with usability as a key factor. The board was oriented to ensure operation of the joystick would not hinder view of the screen. To increase awareness of left handed gamers, the board was designed to be played with the left hand, hence the joystick placement. There is also space on the back of the board for the player to place their hand during gameplay. An image of the board's front and back can be seen in figures 3 and 4 respectively.

5.2 Software

5.2.1 Overview

The overall software flow diagram for "Let the ball drop" can be seen in figure 5 below. This diagram is used to show higher level operation of the software but does not show the lower level operations such as variables and smaller sub functions. This flow chart was created prior to software development, and used as reference throughout the process. There are three main sections of the flowchart, determined by their colour, the green is setup and introduction to the game, purple is the main game loop, and orange is for end of game and high score entry.

The first step in the setup section is to initialize the display and create the game screen platforms in memory. There is then a series of 3 screens which prepare the user for gameplay, each navigated using the joystick to advance. The first screen is the start page, which displays game title and developer name. Following this screen are instructions for gameplay, as well as a short back story to fully immerse the player. Finally before gameplay there is a 5 second countdown to prepare the user, after which the main game loop begins. The start screen, instructions, and countdown screens can be seen in figures 6, 7, 8 respectively.

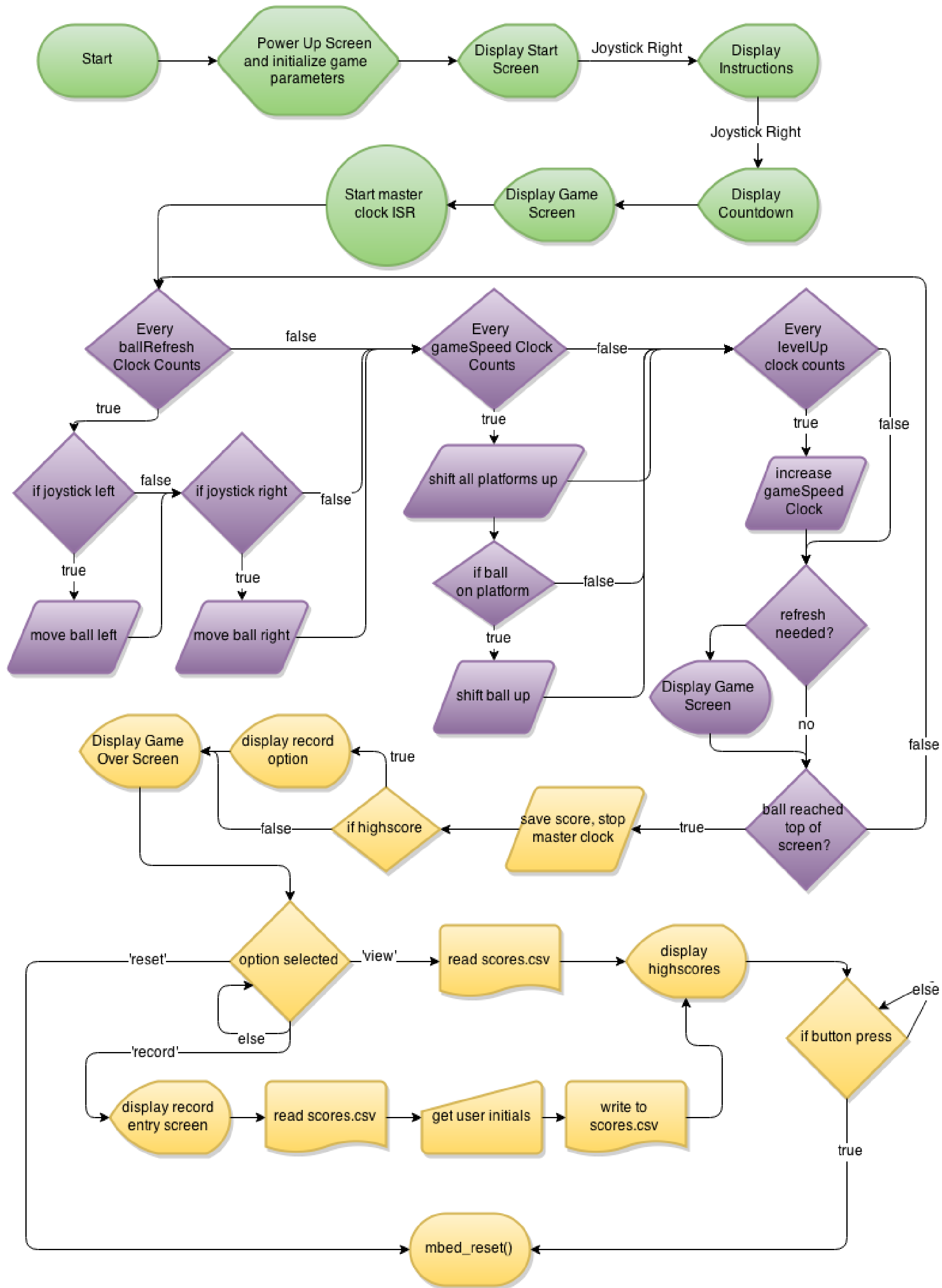


Figure 5: 'Let The Ball Drop' software flow chart.

Figure 6: Introduction screen, showing title and author.

Figure 7: Instruction screen.

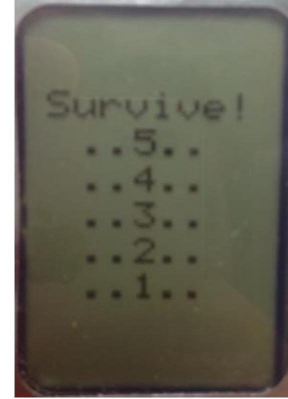


Figure 8: Pre-game Countdown screen.

The main loop, shown in purple, is where the user actually plays the game. The key aspect of this main loop is the use of the master game clock initialized in the setup section. This clock is what dictates the refresh rate of the screen, platform and ball movement, and player score. Instead of using a separate ISR for each animation, a single clock method was chosen to reduce the processing needs of multiple interrupts. The main loop ISR keeps count of number of clock cycles and each animation is triggered after a set number of clock cycles. A screen refresh is only called for if there is a change in the screen buffer. This main loop continues until the player falls to reach a gap in time and is dragged to the top of the screen. The user also has the option to pause the game and display an options menu to toggle sound, restart mBed, or resume game. The main screen display and pause menu can be seen in figure 9 and 10 respectively.

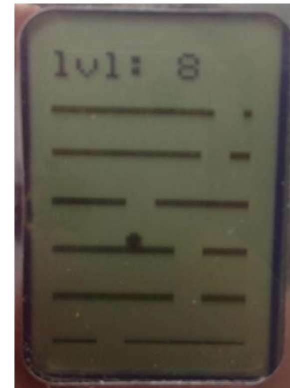


Figure 9: main game loop screen.

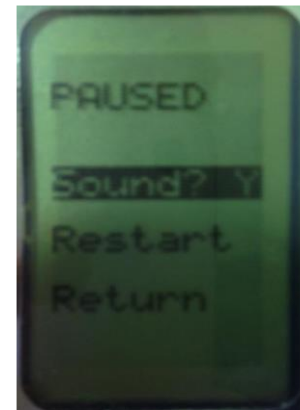


Figure 10: Pause game screen with options!

One of the big features of “Let the Ball Fall” is the ability to record high scores, making the game rewarding and exciting. After the main loop, the program progresses onto the end game sequence shown in orange. First the score is saved and the master clock is detached, then the high scores from ‘scores.csv’ is parsed and checked against the player score. If the player score is higher than any of the three high scores than the player has the option to record their score. The end screen options are displayed, and wait for a user selection before moving on. The user can navigate the options using the up/down on joystick and selecting using the joystick button, the options screen can be seen in figure 11. Further details on this navigation can be found in the function definitions in section 5.2.2. The user can choose to record their high score, view current high scores, or reset the mBed. Selecting ‘view’ takes the user to the view high scores page which lists the top 3 scores from ‘scores.csv’, shown in figure 12 .By choosing to record a high score the player is taken to the initial entry page where they can scan letters and select their initials to be submitted, shown in figure 13.

Figure 11: game over screen with score displayed and options.

Figure 12: high score screen.

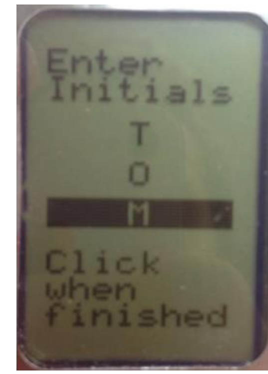


Figure 13: player initial entry screen.

Control of the screen is using a custom class ‘GameScreen’ which inherits the library ‘N5110’ written by Craig A. Evans. The GameScreen class is an extension of the standard library which adds various functionality needed for ‘Let the Ball Drop’ gameplay. More specifically the class introduces functions for drawing game objects, displaying game screens, and printing text. The class is designed purely for interaction with the Nokia 5110 screen and does not manage any user interaction or game mechanics. The object keeps track of ball and platform locations which can be edited externally using access functions. To give absolute control of the display to the main program, the display is only updated or cleared when called from the main program.

All game mechanics and user interaction are handled in the main loop, which relies on a GameScreen object to display all results. The main program operation is fully described in the figure 3, where any display functionality is handled by the GameScreen object. One key aspect of the software design was the choice to use a single ‘master clock’ to control timings of all game animations. This was done to reduce the time spent in ISRs and to ensure that all animations are in unison thus reducing the number of screen refreshes needed. Without the single master clock the game was seen to be less fluid due to the overlapping interrupts and processing requirements.

5.2.2 Function Documentation

The following section completely documents each function developed for ‘Let The Ball Drop’ including each argument and returns. The entirety of code can be found in appendix 9.1 along with further comments on code operation and performance. The doxygen documentation and project can also be found online at: https://developer.mbed.org/users/AppleJuice/code/LetTheBallDrop/.

```
GameScreen ( PinName  pwrPin,
             PinName  scePin,
             PinName  rstPin,
             PinName  dcPin,
             PinName  mosiPin,
             PinName  sclkPin,
             PinName  ledPin
           )          [explicit]
```

Create GameScreen object connected to specific pins. Constructor inherited from base class.

Parameters:

- pwr - Pin connected to Vcc on the LCD display (pin 1)
- sce - Pin connected to chip enable (pin 3)
- rst - Pin connected to reset (pin 4)
- dc - Pin connected to data/command select (pin 5)
- mosi - Pin connected to data input (MOSI) (pin 6)
- sclk - Pin connected to serial clock (SCLK) (pin 7)
- led - Pin connected to LED backlight (must be PWM) (pin 8)

void *createAllPlatforms* ( )

Creates All Platforms. Randomly selects location of gap in each platform and adds all to 'allPlatforms' array.

void *displayCountdown* ( )

Displays Countdown. Countdown from 5 to 1 displayed on screen

```
void displayEndScreen  ( int     lvl,
                         int *   cursor,
                         bool    isRecord
                       )
```

Displays Game over Screen. Shows player level, and displays options. Options can be highlighted according to cursor location array.

Parameters:

- lvl – game level to be displayed
- cursor – array of ints used to highlight options
- isRecord – toggle for displaying “record” option.

```
void displayHighScores  ( char *   s_hs0,
                          char *   s_hs1,
                          char *   s_hs2,
                          int *    d_hs
                        )
```

Display High Score Screen. Top three scores displayed.

Parameters:

- s_hs0 – initials of first place player
- s_hs1 - initials of second place player
- s_hs2 - initials of third place player
- d_hs – array of high scores!

void *displayInstructionScreen* ( )

Displays Instruction Screen. Shows game instructions, navigation method and "back story"

void *displayInstructionScreen2* ( )

Display Second instruction Screen. Shows game controls.

void *displayPauseScreen* (int * cursor,
char isSound
)

Display Pause Menu Screen.

Parameters:

- cursor – array of ints used to highlight selection.
- isSound – display in option selection for sound.

void *displayRecordScreen* (int * cursor,
char ltr1,
char ltr2,
char ltr3
)

Display Initial Entry Screen. Screen for user to enter initials.

Parameters:

- cursor – array of ints used to highlight letters
- ltr1 – first letter of initials
- ltr2 – second letter of initials
- ltr3 – third letter of initials

void *displayStartScreen* ( )

Displays The Start Screen. First screen user sees, author name & title

void *drawAllPlatforms* ( )

Draws All Platforms. Draws all platforms from array allPlatforms in object, requires refresh.

void *drawBall* ( )

Draw Ball. Draws ball on screen from playerBall struct in object, requires refresh.

void *drawPlatform* (int x,
int y
)

Draws Platform. Draws a platform with thickness _platThickness at height y and gap x.

Parameters:

- x – location of gap on platform (right hand pixel)
- y – y position of platform

void *eraseAllPlatforms* ( )

Erases All Platforms. Erases all platforms on screen from array allPlatforms in object, requires refresh.

void *eraseBall* ( )

Erases Ball. Erases ball on screen from playerBall struct in object, requires refresh.

void *erasePlatform* (int y)

Draws Platform. Erases a platform with thickness _platThickness at height y.

Parameters:

- y – y position of platform

void *freeAllPlatforms* ( )

Free All Platforms. Frees memory of each platform in 'allPlatforms' array and destroys the array.

int *getBallR* ( )

Returns:

- ball radius

int *getBallX* ( )

Returns:

- ball X coordinate

int *getBallY* ( )

Returns:

- ball Y coordinate

int *getMaxX* ( )

Returns:

- screen max X

int *getMaxY* ( )

Returns:

- screen max Y

int *getPlatGapSize* ( )

Returns:

- platform gap size

int *getPlatThickness* ( )

Returns:

- platform thickness

void *Initialize* ( )

Initialize Game Screen. Calls N5110::init() which powers up display and sets starting brightness initializes ball position and creates platform list.

Platform *nextClosestPlatform* (int y)

Finds next closest platform.

Parameters:

- y – vertical position on screen

Returns:

- Next closest platform to y.

void *printString* (const char * str,
int x,
int y,
bool invert = false
)

Print String. Overrides N5110::printString(*) to allow vertical orientation text. Requires a refresh to display.

Parameters:

- str – string to be printed
- x – x coordinate of string
- y – y coordinate of string
- invert – color inversion toggle (default = false)

void *setBallPos* (int x,
int y
)

Sets Ball Position.

Parameters:

- x - new x coordinate of ball (right side)
- y - new y coordinate of ball (top)

void *shiftAllPlatforms* (int n = 1)

Shifts Platforms Up. Alter array of platforms to shift each platform up n pixels, requires refresh.

Parameters:

- n - Number of pixels to shift platform.

void *advancePlatforms* (int n = 1)

Advance Platforms. Animate platforms shift up n pixels

Parameters:

- n - number of pixels to shift

void *boop* ( )

PWM speaker control function. Makes a boop sound used for collisions with platforms.

void *clockCounter* ( )

ISR - Clock Counter. Increments number of clock cycles.

void *endScreenController* ( )

Game Over Controller. Displays end game screen, and allows user interaction with options via joystick. Also controls option selection and next screens.

void *highScoreEditor* (char * playerInitials,
bool isRecord
)

High Score Editor. Manages highscore file and recordEntryController().

bool *isBallDirClear* (int dir)

Is Ball Direction Clear? Given ball direction determines if path is clear to move.

Parameters:

- dir - direction ball is travelling in (0 for left 1 for right)

Returns:

- 1 if clear 0 if not

bool *isBallFalling* (int bY)

Check if ball falling. Falling if ball is in free space, not on platform

Parameters:

- bY – ball Y coordinate

Returns:

- 1 if falling 0 if not.

bool *isDown* ( )

Is Joystick Down? Reads joystickY value and checks direction

Returns:

- 1 if down 0 if not.

bool *isLeft* ( )

Is Joystick Left? Reads joystickX value and checks direction

Returns:

- 1 if left 0 if not.

bool *isRecord* ( )

is Record? Check if score achieved by player is a record

Returns:

- 1 if gameLevel is a record 0 if not

bool *isRight* ( )

Is Joystick Right? Reads joystickX value and check direction

Returns:

- 1 if right 0 if not.

bool *isUp* ( )

Is Joystick Up? Reads joystickY value and checks direction.

Returns:

- 1 if up 0 if not.

void *pauseScreenController* ( )

Pause Screen Controller. Displays pause screen and allows user interaction to toggle sound, restart, and resume game.

void *powerSave* ( )

Power Saving Mode. Powers down unused peripherals & Ethernet.

void *pwmLow* ( )

Boop Helper function. Sets pin low when called, then starts timer to recall Boop();

char* *recordEntryController* ( )

Record Entry Controller. Displays initial entry screen, allows user to scroll through letters and select initials.

Returns:

- Initials entered by user.

6.0 Testing

One of the most important phases of any project is testing, which insures the performance and quality of a product meets the original specifications set. This is a rather diverse project with large hardware and software components thus both were tested individually before being deemed "complete".

6.1 Hardware Testing

The hardware testing for this project first began in the prototyping phase where circuit used was first constructed on a breadboard. The tests carried out were designed to validate the performance of each device while also gaining a familiarity of the usage. Simple test software's were developed for each input/output device.

The Nokia 5110 screen was tested on breadboard by setting and resetting each individual pixel on the screen to ensure there were no errors. Then a performance test was carried out where a small box was animated around the screen at gradually increasing speeds to determine the practical max animation speed. This test was carried out as a proof of concept for the game as very high refresh rates are demanded at the higher levels.

The joystick and push button were also tested on a breadboard using a simple program which read and displayed values.

The most important hardware testing was of the power supply circuitry which began in week 1 labs. The first buck converter circuit was built on a breadboard and verified using an oscilloscope and voltmeter. This supply was then used to power the mBed, Nokia 5110 screen, and joystick potentiometers to check for any voltage variances or potential issues.

Once all hardware components were prototyped and sufficiently tested, the printed circuit board (PCB) design began. Throughout the process constant checks were done to ensure that all interconnections were correct and each design rule passed. Testing of the PCB was done in three stages, first all interconnects were tested using the voltmeter continuity setting, then the power supply circuitry was soldered and voltage measured, and finally the remaining components were also soldered. The power supply was soldered and tested first to ensure a suitable output voltage was given and too avoid destroying the more valuable components. Once the PCB was successfully assembled various software's were uploaded to the mBed to test each component individually, and then as an entire unit.

One of the most important concepts when designing an embedded system is power consumption. It is important that the embedded system can operate for as long as possible by using the lowest amount of power. This design is running of a 9V alkaline battery which typically have a capacity of 565 mAh [3]. Using a digital multimeter, the current draw of the system was found to be 152 mA in

the main game loop. Since the main loop is the state that the system will spend majority of its time, this is the current value that will be used in calculations. Given the battery capacity and average current draw of the system it can be found that the design should operate for 3.7 hours before the battery needs to be replaced.

6.2 Hardware Refinements

Throughout the prototyping phase various hardware refinements were made to meet the design requirements. On first prototype of the buck converter the output voltage was found to be above the required 5V. Due to an error in discrete component calculations a wrong value capacitor was used in the initial prototype. Due to the simplicity of the hardware involved in the project, the number of refinements was minimal, and once the PCB was designed further refinements were impossible.

One of the longest refinement processes in hardware design is in placing and routing components on the PCB. Throughout the design various changes were made to ensure the board design was ergonomic and practical while meeting all the requirements. The joystick and screen placement were extremely important as if either were difficult to view or access it would drastically hinder gameplay. The joystick was placed to the left of the screen (game designed for left handed players) with space on the reverse side of the PCB for finger placement. This organization allows the player to hold the board in just their left hand as they play. The mBed was also oriented in a way such that the USB port is easily accessed for uploading new code revisions.

6.3 Software Testing

Software testing is important because everyone makes mistakes, some can be harmless but many can lead to significant issues down the line for software developers. In development of 'Let the Ball Fall' strict testing methodologies were followed and new functionality was only added to the code when sufficient testing was carried out. Each function was tested with an extreme case, a null case and an expected case to ensure appropriate results were returned. Due to the small scale of the software, the tests were not automated and were manually created using small programs dedicated to testing each individual function. Some testing time was also spent to ensure that allocated pointers could be accessed in all needed areas of the programs and memory was not freed pre maturely.

When each individual function was appropriately tested, the main game loop was then developed. With each new function revision the software would be tested again to ensure a consistent and reliable program. By testing throughout the process minimal time was needed after development to debug program operation.

Once the main game loop was complete the embedded system was then passed through various stages of alpha and then beta testing to reach the final product submitted for ELEC 2645.

6.4 Software Refinements

Throughout the entire development process constant refinements were made to the code to minimize memory usage as well as code readability. During each function testing the code was analysed and improved to insure optimal performance while maintaining good programming practices. Majority of the software refinements occurred during the alpha and beta testing periods, to improve usability and game feel.

The alpha testing was carried out by the sole developer and entailed testing gameplay and game screens to find any possible bugs or glitches. During this phase the tester aims were to 'break' the game by doing unexpected actions and thoroughly testing the collision detection algorithm, controller sensitivity, and interrupt timing. The alpha phase lasted from the start of development to the point at which a nearly complete project was achieved.

After alpha testing the embedded system was then left in the hands of a group of beta testers which were asked to simply play the game and recommend improvements. This phase led to changes in animation timings, control sensitivity, and tweaks to the option selection menu. The beta phase was used to put the final touches on the game and tweak game mechanics for a more enjoyable experience.

7.0 Conclusion

After four weeks of labs and six weeks of development, 'Let the Ball Drop' was completed and surpassed many expectations. The small handheld gaming system successfully created an addictive and enjoyable gaming experience. Of each of the thirteen specifications listed all were met, with the exception of the idea to have a separate ISR for each animation which was refined to a single ticker during software testing.

Overall, 'Let the Ball Drop' was a great success from a hardware and software aspect. The hardware design proved to be ergonomic in design, with a great layout and no issues tested so far. The electrical manufacturing process of SMD soldering proved to be a challenging but rewarding task, resulting in a small and effective PCB design. The software developed for this project is also very strong, featuring object oriented design practices with a minimal memory imprint and concise understandable code. The software is well commented and explained and currently in an ideal state for any further improvements and feature upgrades.

The first step in developing this embedded system was to generate a set of aims that were broad statements used to guide progress in development. The embedded system has a small form factor, good battery life, ergonomic and usable PCB layout, enjoyable intuitive gameplay, and is extremely addicting to play. Constantly referring back to these basic aims allows the development to not only run smoothly but also stay on track to meeting the initial game vision.

From the objectives and constraints rose the project specifications which were used as the framework for this project and key points throughout development. On the hardware side the device is powered by a 9V battery using a 9-5V buck converter for regulation, which has been a successful decision. The Nokia 5510 screen and joystick were great choices for display and input, however if a TFT LCD screen were available the colour functionality would add to the user experience [5]. The decision to inherit and extend on Craig A. Evans N5510 library proved to be a great success, adding another level of functionality to the screen and enabling quick and easy changes to the game screen. Within the GameScreen class, using an array to manage game objects made accessing and editing item easy and understandable in the code. Another option was to use a lower level linked list, but using an array proved to be a stronger choice for code readability, with minimal memory side effects. The progression of screens at launch allow the user to become acquainted with the controls and more immersed in the gameplay. With the in-game options screen introduced the embedded system is now usable in public environments as sound can be toggled and game paused. Failing to stay below the top of screen during gameplay brings up the game over screen, which allows the user to save their score or compare it to

other scores recorded. This addition is what helped achieve the aim of creating an addictive game, without it there is no competitive edge and players would get bored of the gameplay [6]. The only change from specification to production was in the decision to use a single master game clock instead of multiple dedicated timers for each animation. The choice was made to ensure that the aim "To display game graphics in a fluent manner" was met; the dedicated timers led to jumpy timings and excessive screen refresh calls.

With more time and resources the hardware design would be shrunk down to make the embedded system more portable and robust. It would also be advisable for an enclosure to be created to protect the components inside. Another potential hardware improvement would be to remove the mBed LPC 1768 microcontroller and rebuild the circuit directly on the board. This improvement would greatly reduce the form factor of the PCB but make code revisions impossible, the change should only be made if pushing to production. The software should also be further tested and should go through a much longer testing and refinement phase than what is allowed in this project. The object structure of the software is also not complete, separate classes for both platforms and the game ball should be created. More effort should also be made to concealing the 'scores.csv' file stored on the mBed, some encryption should be introduced to protect the contents.

8.0 References

[1]M. Barr, A. Massa and M. Barr, Programming embedded systems. Sebastopol, Calif.: O'Reilly, 2006.

[2] Developer.mbed.org, 'mbed LPC1768 | mbed', 2015. [Online]. Available: https://developer.mbed.org/platforms/mbed-LPC1768/. [Accessed: 04- May- 2015].

[3]E. Olivetti, J. Gregory and R. Kirchain, 'LIFE CYCLE IMPACTS OF ALKALINE BATTERIES WITH A FOCUS ON END-OF-LIFE', Massachusetts Institute of Technology, Massachusetts, 2015.

[4]S. Heble, 'What is a microcontroller? And how does it differ from a microprocessor? - maxEmbedded', maxEmbedded, 2011. [Online]. Available: http://maxembedded.com/2011/06/mcu-vs-mpu/. [Accessed: 08- May- 2015].

[5] Sitronix, "262K Color Single-Chip TFT Controller/Driver" ST7735R datasheet, 2009

[6]K. Young, 'Understanding Online Gaming Addiction and Treatment Issues for Adolescents', The American Journal of Family Therapy, vol. 37, no. 5, pp. 355-372, 2009.

[7]M. Barr, 'Introduction to Pulse Width Modulation (PWM) | Embedded Systems Experts',Barrgroup.com, 2015. [Online]. Available: http://www.barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation. [Accessed: 04- May- 2015].

[8] Phillips Semiconductors, "48 x 84 pixels matrix LCD controller/driver", PCD8544 datasheet, Apr. 1999.

9.0 Appendix

9.1.0 Code

9.1.1 Main.h

```
/**
@file main.h
@brief Header file for main game containing function prototypes, namespaces
and global variables.
@autor Thomas Davies
@date May 2015
*/

#ifndef main_H
#define main_H

#include "mbed.h"
#include "GameScreen.h"
#include "PowerControl/PowerControl.h"
#include "PowerControl/EthernetPowerControl.h"

/**
@namespace serial
@brief serial communication with host
*/
Serial serial(USBTX,USBRX);

/**
@namespace lcd
@brief game screen controller
*/
GameScreen lcd(p7,p8,p9,p10,p11,p13,p21);

/**
@namespace leds
@brief Output for onboard LEDs
*/
BusOut leds(LED4,LED3,LED2,LED1);

/**
@namespace joystickX
@brief input from joystick X, on p15
*/
AnalogIn joystickX(p15);

/**
@namespace joystickY
@brief input from joystick Y, on p16
*/
AnalogIn joystickY(p16);

/**
@namespace joystickButton
@brief input from joystick button, on p17
*/
DigitalIn joystickButton(p17);
```

```
/**
@namespace buzzer
@brief digital output to buzzer on p5
*/
DigitalOut buzzer(p5);

/**
@namespace local
@brief local file system
*/
LocalFileSystem local("local");

/**
@namespace gameClock
@brief the master clock of game operation. All animation timings are based on
this clock.
*/
Ticker gameClock;

/**
@namespace lowFlipper
@brief used for PWM speaker control, triggers pin low.
*/
Timeout lowFlipper;

/**
@namespace highFlipper
@brief used for PWM speaker control, triggers pin high.
*/
Timeout highFlipper;

//#############GLOBAL VARIABLES#################//

int clockCount = 0;          ///< master clock counter
bool isFirstCheck = true;    ///< first check in clock cycle?
bool isFirstHit = true;      ///< first hit on a platform?
bool isFirstSpeedUp = true; ///< first game speed up?
int gameLevel = 0;           ///< current game level
int gameSpeed = 41;          ///< current game speed
int numPixelsToJump = 1;     ///< number of pixels ball/plat moves per
animation
int numPlatformShifts = 0;  ///< number of times platforms shifted, used for
score calculation
int pwmCount = 1000;         ///< pwm count for platform collision sound
char isSound = 'Y';          ///< sound toggle, 'Y' for sound enabled.

//############FUNCTION PROTOTYPES#################//

/** Reset Mbed */
extern "C" void mbed_reset();

/** Advance Platforms
*
* animate platforms shift up n pixels
*
* @param n - number of pixels to shift
*/
```

```
void advancePlatforms(int n=1);

/** Check if ball falling
*
* Falling if bal is in free space, not on platform
* @param bY - ball Y coordinate
* @return 1 if falling 0 if not.
*/
bool isBallFalling(int bY);

/** Is Joystick Right?
*
* reads joystickX value and check direction
*@returns 1 if right 0 if not.
*/
bool isRight();

/** Is Joystick Left?
*
* reads joystickX value and check direction
*@returns 1 if left 0 if not.
*/
bool isLeft();

/** Is Joystick Up?
*
* reads joystickY value and check direction
*@returns 1 if up 0 if not.
*/
bool isUp();

/** Is Joystick Down?
*
* reads joystickY value and checks direction
*@returns 1 if down 0 if not.
*/
bool isDown();

/** ISR - Clock Counter
*
*increments number of clock cycles
*/
void clockCounter ();

/** Is Ball Direction Clear?
*
*given ball direction determines if path is clear to move.
*@param dir - direction ball is travelling in (0 for left 1 for right)
*@returns 1 if clear 0 if not
*/
bool isBallDirClear(int dir);

/** Game Over Controller
*
* Displays end game screen, and allows user interaction with options via
joystick.
* Also controls option selection and next screens.
```

```
*/
void endScreenController();

/** Record Entry Controller
*
* Displays initial entry screen, allows user to scroll through letters and
select initials.
* @return intitials entered by user.
*/
char* recordEntryController();

/** is Record?
*
* check if score achieved by player is a record
* @return 1 if gameLevel is a record 0 if not
*/
bool isRecord ();

/** High Score Editor
*
* Manages highscore file and recordEntryController().
*/
void highScoreEditor(char *playerInitials,bool isRecord);

/** Pause Screen Controller
*
* Displays pause screen and allows user interaction to toggle sound,restart,
and resume game.
*/
void pauseScreenController();

/** PWM speaker control function
*
* makes a boop sound used for collisions with platforms.
*/
void boop ();

/** Boop Helper function
*
* sets pin low when called, then starts timer to recall Boop();
*/
void pwmLow();

/** Power Saving Mode
*
* Powers down unused peripherals & ethernet
*/
void powerSave();


#endif
```

9.1.2 Main.cpp

```
/**
@file main.cpp
```

```
@brief Source file for main game containing function defintions and main
loop.
@autor Thomas Davies
@date May 2015
*/

#include "main.h"


int main()
{

    lcd.Initialize();
    powerSave();

    lcd.displayStartScreen();
    while(!isRight()) {}       //wait until right button pressed

    wait(0.5);          //just to ignore layover from first button press.

    lcd.displayInstructionScreen();
    while(!isRight()) {}    //wait until right button pressed.

    wait(0.5);
    lcd.displayInstructionScreen2();
    while(!isRight()) {}    //wait until right button pressed.

    lcd.displayCountdown();
    lcd.clear();

    bool refresh = false;

    lcd.drawAllPlatforms();
    lcd.drawBall();
    lcd.refresh();

    gameClock.attach(&clockCounter,0.003);  //attach the master clock!

    //MAIN GAME LOOP
    //multiple runs on a single loop. stop with a bool

    char buffer[10];
    while(true) {
        if (isFirstCheck) { //if first time checking this clock cycle.
            if(clockCount %(gameSpeed) == 0) {
                //advance platforms
                advancePlatforms(numPixelsToJump);
                numPlatformShifts ++;

                if (numPlatformShifts%5 == 0)
                {
                    gameLevel ++;
                 }
                refresh = true;
            }

            if(clockCount%14 == 0) {
```

```
                //ball movement
                if (isLeft() && isBallDirClear(1)) {    //if josystick left 
and direction is clear
                    lcd.eraseBall();
                    lcd.setBallPos(lcd.getBallX()+2,lcd.getBallY());
                    refresh = true;

                } else if (isRight() && isBallDirClear(0)) {
                    lcd.eraseBall();
                    lcd.setBallPos(lcd.getBallX()-2,lcd.getBallY());
                    refresh = true;
                }
            }

            //if the ball is in free space, animate it falling!
            if (clockCount%(10/numPixelsToJump) == 0) {
                if(isBallFalling(lcd.getBallY())) {
                    lcd.eraseBall();
                    lcd.setBallPos(lcd.getBallX(),lcd.getBallY() + 
numPixelsToJump);
                    refresh = true;
                    isFirstHit = true;
                }
            }

            if ((clockCount%(-45*gameSpeed + 2045) == 0)) {    //starting 
levels are quick to progress, later slower.
                if (isFirstSpeedUp) //skip first accepted modulus!
                    isFirstSpeedUp = false;
                else {
                    gameSpeed --;
                    isFirstSpeedUp = true;
                }
            }

            if (joystickButton.read())      //pause menu!
            {
                int oldClock = clockCount;  //save the clock count!
                pauseScreenController();
                clockCount = oldClock;
                wait(0.3);
                lcd.clear();
                refresh = true;
            }

            if (refresh) {
                lcd.drawBall();
                lcd.drawAllPlatforms();

                sprintf(buffer,"lvl:%d",gameLevel);
                lcd.printString(buffer,lcd.getMaxX() - 2,-1);
                lcd.refresh();
                refresh = false;
            }


            //check if ball hit roof
```

```
            if (lcd.getBallY() == 7) {
                break;
            }


            isFirstCheck = false;
        }
    }

    gameClock.detach();

    lcd.clear();

    endScreenController();
    return 1;
}


//############FUNCTION DEFINITIONS################//
void advancePlatforms(int n)
{
    //ADVANCE
    lcd.eraseAllPlatforms();
    lcd.shiftAllPlatforms(n);
    lcd.drawAllPlatforms();

    //if ball is on platform
    if(!isBallFalling(lcd.getBallY()-n)) {
        //auditory feedback if first collision
        if (isFirstHit) {
            boop();
            isFirstHit = false;
        }
        //move ball up with platform
        lcd.eraseBall();
        lcd.setBallPos(lcd.getBallX(),lcd.getBallY() -n);
    }


}

bool isBallFalling(int bY)
{
    int bX = lcd.getBallX();    //ball X pos
    int bR = lcd.getBallR();    //ball radius

    //find next platform lower than ball
    //check Y pos to see if ball is on platform, return false if on plat
    //check X pos to see if ball is over gap, return true if over gap
    Platform closestPlatform = lcd.nextClosestPlatform(bY - bR);


    //if bottom of ball on level of platform
    if ((bY+bR) == (closestPlatform.y)) {
        if (bX > closestPlatform.x && (bX+bR-1) < (closestPlatform.x +
lcd.getPlatGapSize())) { //OVER GAP
            return true;
```

```
        } else {
            return false;
        }      //On platform!
    } else if ((bY+bR) >= (lcd.getMaxY()-1)) {
        return false;
    } else  //FREE SPACE
        return true;
}


bool isBallDirClear(int dir)
{
    Platform closestPlatform = lcd.nextClosestPlatform(lcd.getBallY() - 
lcd.getBallR()+2);

    int ballRight = lcd.getBallX();
    int ballLeft = ballRight+lcd.getBallR();
    int ballTop = lcd.getBallY();
    int ballBottom = ballTop + lcd.getBallR();

    if (dir == 1) {
        if (ballBottom > closestPlatform.y)    //is ball inside a gap in 
platform?
            if (ballLeft >= (closestPlatform.x+lcd.getPlatGapSize()))    //is 
the ball next to gap wall?
                    return false;
        if (ballLeft < lcd.getMaxX()) {    //is ball next to left hand wall?
            return true;
        }
    } else if (dir == 0) {
        if (ballBottom > closestPlatform.y)    //is ball inside a gap in 
platform?
            if (ballRight <= closestPlatform.x)
                return false;

        if (ballRight > 0) {   //right hand wall check
            return true;
        }
    }
    return false;
}

bool isRight()
{
    double count = 0.0;
    double total = 0.0;
    //obtain an average to negate outliers
    while (count < 10) {
        count ++;
        total += joystickX.read();
    }
    if ((total/count)> 0.8)
        return true;

    return false;
}
```

```
bool isLeft()
{
    double count = 0.0;
    double total = 0.0;
    //obtain an average to negate outliers
    while (count < 10) {
        count ++;
        total += joystickX.read();
    }
    if ((total/count) <  0.2)
        return true;

    return false;
}

bool isUp()
{
    double count = 0.0;
    double total = 0.0;
    //obtain an average to negate outliers
    while (count < 10) {
        count ++;
        total += joystickY.read();
    }

    if ((total/count) <  0.2)
        return true;

    return false;
}
bool isDown()
{
    double count = 0.0;
    double total = 0.0;
    //obtain an average to negate outliers
    while (count < 10) {
        count ++;
        total += joystickY.read();
    }
    if ((total/count) >  0.8)
        return true;

    return false;
}

void clockCounter ()
{
    //increment clock count!
    clockCount ++;
    isFirstCheck = true;
}

void endScreenController()
{
    int cursorLoc = 0;
    //cursor location array.
    int cursorArray[3][3] = {
```

```
        {1,0,0},
        {0,1,0},
        {0,0,1}
    };

    //check if user achieved high score
    bool record = isRecord();

    //if not a record move mouse start to second option
    if (!record)
        cursorLoc = 1;

    lcd.displayEndScreen(gameLevel,cursorArray[cursorLoc],record);
    bool change = false;

    while (true) {
        if (isUp()) {
            cursorLoc ++;   //advance cursor menu
            change = true;
        } else if (isDown()) {
            cursorLoc --;   //move back cursor menu
            change = true;
        }

        //cyclical cursor management
        if (cursorLoc == 3)
            cursorLoc = 1-record;
        else if (cursorLoc == (-record))
            cursorLoc = 2;

        //if theres a change to buffer, show it
        if (change) {
            lcd.displayEndScreen(gameLevel,cursorArray[cursorLoc],record);
            wait(0.5);
            change = false;
        }

        //menu selection manager
        if (joystickButton.read()) {
            switch(cursorLoc) {
                case 0:
                    //take to initial entry page! (only allow if score > top
3)
                    wait(0.3);
                    highScoreEditor(recordEntryController(),record);

                    wait(1);
                    //reset once button pressed!
                    while(!joystickButton.read()) {}
                    mbed_reset();
                    break;
                case 1:
                    //take to high score page!
                    highScoreEditor("",0);
                    wait(1);
                    //reset once button pressed.
                    while(!joystickButton.read()) {}
```

```
                    mbed_reset();
                    break;
                case 2:
                    //restart
                    mbed_reset();
                    break;
            }
        }
    }
}

char* recordEntryController()
{
    //array of letters to traverse
    char *letters = {"ABCDEFGHIJKLMNOPQRSTUVWXYZ"};

    //array of ints, each corresponding to letter in letters array
    int ptr[3] = {0,0,0};

    //cursor management variables
    int cursorLoc = 0;
    int cursorArray[3][3] = {
        {1,0,0},
        {0,1,0},
        {0,0,1}
    };


lcd.displayRecordScreen(cursorArray[cursorLoc],letters[ptr[0]],letters[ptr[1]
],letters[ptr[2]]);

    bool change = false;

    while (true) {
        if (isUp()) {
            cursorLoc ++;
            change = true;
        } else if (isDown()) {
            cursorLoc --;
            change = true;
        } else if (isLeft()) {
            //cyclical letter management
            if (ptr[cursorLoc] == 0)
                ptr[cursorLoc] = 25;   //last letter
            else
                ptr[cursorLoc] --;  //previous letter
            change = true;
        } else if (isRight()) {
            if (ptr[cursorLoc] == 25)
                ptr[cursorLoc] = 0;  //first letter
            else
                ptr[cursorLoc] ++;  //next letter
            change = true;
        }

        //cyclical mouse management
        if (cursorLoc == 3)
```

```
            cursorLoc = 0;
        else if (cursorLoc == -1)
            cursorLoc = 2;

        if (change) {

lcd.displayRecordScreen(cursorArray[cursorLoc],letters[ptr[0]],letters[ptr[1]
],letters[ptr[2]]);
            wait(0.2);
            change = false;
        }

        if (joystickButton.read()) {
            //go to high score screen
            static char initials[10];

sprintf(initials,"%c%c%c",letters[ptr[0]],letters[ptr[1]],letters[ptr[2]]);
            return initials;
        }
    }
}

void highScoreEditor(char *playerInitials,bool isRecord)
{
    //check if lvl is in top 3
    FILE *fp = fopen("/local/scores.csv","r");  //open for reading.

    char s_hs0[3],s_hs1[3],s_hs2[3];
    int d_hs[3];

    //if file exists, parse it
    if (fp != NULL) {

        char buffer[2];

        fscanf(fp,"%3s,%s",s_hs0,buffer);
        d_hs[0] = atoi(buffer);
        fscanf(fp,"%3s,%s",s_hs1,buffer);
        d_hs[1] = atoi(buffer);
        fscanf(fp,"%3s,%s",s_hs2,buffer);
        d_hs[2] = atoi(buffer);

        fclose(fp);
    }
    //else set arbitrary values!
    else {
        d_hs[0] = 0;
        d_hs[1] = 0;
        d_hs[2] = 0;

        strcpy(s_hs0,"---");
        strcpy(s_hs1,"---");
        strcpy(s_hs2,"---");
    }

    if (isRecord) {
```

```
        if (gameLevel >= d_hs[2]) {
            d_hs[2] = gameLevel;
            strncpy(s_hs2,playerInitials,3);
        }
        if (gameLevel >= d_hs[1]) {
            d_hs[2] = d_hs[1];
            d_hs[1] = gameLevel;
            strncpy(s_hs2,s_hs1,3);
            strncpy(s_hs1,playerInitials,3);
        }
        if (gameLevel >= d_hs[0]) {
            d_hs[1] = d_hs[0];
            d_hs[0] = gameLevel;
            strncpy(s_hs1,s_hs0,3);
            strncpy(s_hs0,playerInitials,3);
        }

        char buffer[40];

sprintf(buffer,"%s,%d\n%s,%d\n%s,%d",s_hs0,d_hs[0],s_hs1,d_hs[1],s_hs2,d_hs[2
]);
        fp = fopen("/local/scores.csv","w");
        fprintf(fp,buffer);
        fclose(fp);
    }

    lcd.displayHighScores(s_hs0,s_hs1,s_hs2,d_hs);
}

bool isRecord ()
{
    //check if lvl is in top 3
    FILE *fp = fopen("/local/scores.csv","r");  //open for reading.

    //if file doesn't exist, then player has a highscore!
    if (fp == NULL)
        return true;

    int highScore;
    char buffer[2];
    char temp[3];

    for (int i = 0; i < 3; i++) {
        fscanf(fp,"%3s,%s",temp,buffer);
        highScore = atoi(buffer);
        if (gameLevel > highScore) {
            fclose (fp);
            return 1;
        }
    }
    fclose(fp);
    return 0;
}

void boop ()
{
    if (isSound == 'Y')
```

```
    {
        buzzer.write(1);
        lowFlipper.attach_us(&pwmLow,pwmCount);
    }
}

void pwmLow()
{
    if (pwmCount != 4000)
    {
        buzzer.write(0);
        highFlipper.attach_us(&boop,pwmCount);
        pwmCount += 50;
   }
   else
    pwmCount = 1000;

}

//can toggle sound or quit game.
void pauseScreenController()
{
    int cursorLoc = 0;
    int cursorArray[3][3] = {
        {1,0,0},
        {0,1,0},
        {0,0,1}
    };

    lcd.displayPauseScreen(cursorArray[cursorLoc],isSound);

    bool change = false;
    bool resume = false;

    while (!resume) {
        if (isUp()) {
            cursorLoc ++;
            change = true;
        } else if (isDown()) {
            cursorLoc --;
            change = true;
        } else if (isLeft() || isRight()) {
            if (isSound == 'Y')
                isSound = 'N';
            else
                isSound = 'Y';
            change = true;
        }

        //cyclical mouse management
        if (cursorLoc == 3)
            cursorLoc = 0;
        else if (cursorLoc == -1)
            cursorLoc = 2;

        if (change) {
            lcd.displayPauseScreen(cursorArray[cursorLoc],isSound);
```

```
            wait(0.2);
            change = false;
        }

        if (joystickButton.read()) {
            switch (cursorLoc) {
                case 1:
                    mbed_reset();
                    break;
                case 2:
                    resume = true;
                    break;
                default:
                    break;
                    }
        }
    }
}

void powerSave()
{
        PHY_PowerDown();    //power down ethernet
        //mbed automatically manages peripheral power, no need to manually
disable each
}
```

9.1.3 GameScreen.h

```
/**
@file GameScreen.h
@brief Simple library for managing "Let The Ball Drop" Game screen
@brief extends on N5110 library created by Craig.A.Evans
```

```
@brief Adds functionality to animate game platforms and ball,
@brief also displays various game screens throughout gameplay.
@brief printString function from base class N5110 is overwritten and
@brief extened to vertical orientation text and inverted pixel scheme.
@author Thomas Davies
@date May 2015
*/


#ifndef GameScreen_H
#define GameScreen_H

#include "mbed.h"
#include "N5110.h"

/**
Platform structure
*/
struct Platform {
    int id;     ///< id identifier of platform
    int x;      ///< x-location of gap in platform
    int y;      ///< y-location of top pixel in platform
};

/**
Ball structure
*/
struct Ball {
    int x;      ///< ball x-location (far right pixel)
    int y;      ///< ball y-location (top pixel)
};

/** GameScreen Class
*
*Nokia 5110 screen controller with specific functions for manageing game
objects and screens.
*
*/
class GameScreen: public N5110::N5110
{
public:

    /** Create GameScreen object connected to specific pins
    *
    * constructor inhereted from base class
    * @param pwr Pin connected to Vcc on the LCD display (pin 1)
    * @param sce Pin connected to chip enable (pin 3)
    * @param rst Pin connected to reset (pin 4)
    * @param dc  Pin connected to data/command select (pin 5)
    * @param mosi Pin connected to data input (MOSI) (pin 6)
    * @param sclk Pin connected to serial clock (SCLK) (pin 7)
    * @param led Pin connected to LED backlight (must be PWM) (pin 8)
    *
    */
    explicit GameScreen(PinName pwrPin, PinName scePin, PinName rstPin,
PinName dcPin, PinName mosiPin, PinName sclkPin, PinName ledPin)
        :N5110(pwrPin, scePin,rstPin,dcPin,mosiPin,sclkPin,ledPin) {}
```

```
    /** Initialise Game Screen
    *
    *   Calls N5110::init() which powers up display and sets starting
brightness
    *   initialises ball position and creates platform list.
    *
    */
    void Initialize();

    /** Print String
    *
    * Overrides N5110::printString(*) to allow vertical orientation text.
    * requires a refresh to display.
    *
    *@param str - string to be printed
    *@param x - x coordinate of the string
    *@param y - y coordinate of the string
    *@param invert - color inversion toggle (default = false)
    *
    */
    void printString(const char * str,int x,int y,bool invert = false);

    /** Draw Ball
    *
    * Draws ball on screen from playerBall struct in object, requires
refresh.
    *
    */
    void drawBall();

    /** Erases Ball
    *
    * Erases ball on screen from playerBall struct in object, requires
refresh.
    *
    */
    void eraseBall();

    /** Draws All Platforms
    *
    * Draws all platforms from array allPlatforms in object, requires
refresh.
    *
    */
    void drawAllPlatforms();

    /** Erases All Platforms
    *
    * Erases all platforms on screen from array allPlatforms in object,
requires refresh.
    *
    */
    void eraseAllPlatforms();

    /** Shifts Platforms Up
    *
```

```
    * Alter array of platforms to shift each platform up n pisxels, requires
refresh.
    *@param n - number of pixels to shift platform.
    *
    */
    void shiftAllPlatforms(int n=1);

    /** Finds next closest platform
    *
    *@param y - vertical position on screen
    *@return next closest platform to y.
    *
    */
    Platform nextClosestPlatform(int y);

    /** Displays The Start Screen
    *
    * first screen user sees, author name & title
    *
    */
    void displayStartScreen();

    /** Displays Instruction Screen
    *
    * shows game instructions, navigation method and "back story"
    *
    */
    void displayInstructionScreen();

    /** Displays Countdown
    *
    * countdown from 5 to 1 displayed on screen.
    *
    */
    void displayCountdown();

    /** displays Game Over Screen
    *
    * shows player level, and displays options.
    * options can be highlighted according to cursor location array
    * @param lvl - game level to be displayed.
    * @param cursor - array of ints used to highlight options.
    * @param isRecord - only display record option if player achieved a
record
    *
    */
    void displayEndScreen(int lvl,int *cursor,bool isRecord);

    /** Display Initial Entry Screen
    *
    * screen for user to enter initials.
    *
    *@param cursor - array of ints used to highlight letters
    *@param ltr1 - first letter of initials
    *@param ltr2 - second letter of initials
    *@param ltr3 - third letter of initials
    *
```

```
    */
    void displayRecordScreen(int *cursor, char ltr1, char ltr2,char ltr3);

    /** Display High Score Screen
    *
    *@param s_hs0 - initials of top player.
    *@param s_hs1 - initials of second player.
    *@param s_hs2 - initials of third player.
    *@param d_hs - array of high score levels.
    */
    void displayHighScores(char *s_hs0,char *s_hs1,char *s_hs2,int *d_hs);

    /** Display Second instruction Screen
    *
    */
    void displayInstructionScreen2();

    /** Display Pause Menu Screen
    *
    *@param cursor - array of ints used to highlight selection
    *@param isSound - toggle for "yes" or "no" text for sound
    *
    */
    void displayPauseScreen(int *cursor, char isSound);

    /** Sets Ball Position
    *
    *@param x - new x coordinate of ball (right side)
    *@param y - new y coordinate of ball (top)
    */
    void        setBallPos(int x, int y)        { playerBall.x = x;
playerBall.y = y; }

    //ACCESSOR FUNCTIONS
    /** @return platform gap size */
    int         getPlatGapSize()                { return platGapSize_;
}

    /** @return screen max Y */
    int         getMaxY()                          { return maxY_;
}

    /** @return screen max X */
    int         getMaxX()                          { return maxX_;
}

    /** @return platform thickness */
    int         getPlatThickness()                 { return platThickness_;
}

    /** @return ball radius */
    int         getBallR()                      { return ballRadius_;       }

    /** @return ball X coordinate */
    int         getBallX()                      { return playerBall.x;      }

    /** @return ball Y coordinate*/
```

```
    int         getBallY()                          { return playerBall.y;      }

    /** Creates All Platforms
    *
    * Randomly selects location of gap in each platform and adds all to
'allPlatforms' array.
    *
    */
    void createAllPlatforms();

   /** Draws Platform
   *
   * Draws a platform with thickness _platThickness at height y and gap x.
   *
   *@param x - location of gap on platform (right hand pixel)
   *@param y - y position of platform
   *
   */
    void drawPlatform(int x,int y);

    /** Draws Platform
    *
    * Erases a platform with thickness _platThickness at height y.
    *
    *@param y - y position of platform
    *
    */
    void erasePlatform(int y);

    /** Free All Platforms
    *
    * frees memory of each platform in 'allPlatforms' array and destroys the
array.
    *
    */
    void freeAllPlatforms();

    static const int platGapSize_ = 8;          ///< standard platform gap
width in pixels
    static const int platThickness_ = 2;        ///< platform thickness in
pixels
    static const int platSpacing_ = 14;         ///< subsequent platform
spacing in pixels
    static const int maxX_ = 48;                ///< maximun horizontal pixel
    static const int maxY_ = 84;                ///< maximum vertical pixel
    static const int ballRadius_ = 4;           ///< size of player ball
    static const int numPlatforms_ = 6;         ///< total number of
platforms
    Ball playerBall;                            ///< player ball structure
    Platform *allPlatforms[numPlatforms_];      ///< array used to track each
platform, and refresh when needed!
};


#endif
```

9.1.4 GameScreen.cpp

```
#include "mbed.h"
#include "N5110.h"
#include "GameScreen.h"

//initialise function, sets ball pos and creates platforms
```

```
void GameScreen::Initialize()
{
    init();         //power up screen and set brightness

    //set ball pos
    playerBall.x = maxX_/2;
    playerBall.y = maxY_ - (int)(maxY_/3 * 2) - 5;

    createAllPlatforms();
}

//draw platform
//   ___________  ________  < y
//                ^ x
void GameScreen::drawPlatform(int x,int y)
{
    for (int a = 0; a < 48; a ++)
    {
        for (int b = y; b < (y+platThickness_); b++)
        {
            //skip pixels of gap
            if (a > x && a < (x + platGapSize_))
                break;

            //skip pixels below maxY (lets platforms 'emerge' instead of
'appear')
            if (b > (maxY_ - 1))
                break;

            setPixel(b,a);
        }
    }
}

//erase platform
void GameScreen::erasePlatform(int y)
{
    for (int a = 0; a < 48; a ++)
    {
        for (int b = y; b < (y+platThickness_); b++)
        {
            //skip pixels below maxY (lets platforms 'emerge' instead of
'appear')
            if (b > (maxY_ - 1))
                break;

            clearPixel(b,a);
        }
    }
}

//draw the player ball where (x,y) is top right corner.
//      XXP <-- P(x,y)
//     XXXX
//     XXXX
//      XX
void GameScreen::drawBall()
```

```
{
    //more elegant ways to do this...but this is most efficient
    setPixel(playerBall.y,playerBall.x+1);
    setPixel(playerBall.y,playerBall.x+2);
    setPixel(playerBall.y+1,playerBall.x);
    setPixel(playerBall.y+1,playerBall.x+1);
    setPixel(playerBall.y+1,playerBall.x+2);
    setPixel(playerBall.y+1,playerBall.x+3);
    setPixel(playerBall.y+2,playerBall.x);
    setPixel(playerBall.y+2,playerBall.x+1);
    setPixel(playerBall.y+2,playerBall.x+2);
    setPixel(playerBall.y+2,playerBall.x+3);
    setPixel(playerBall.y+3,playerBall.x+1);
    setPixel(playerBall.y+3,playerBall.x+2);
}

//draw the player ball where (x,y) is top right corner.
void GameScreen::eraseBall()
{
    clearPixel(playerBall.y,playerBall.x+1);
    clearPixel(playerBall.y,playerBall.x+2);
    clearPixel(playerBall.y+1,playerBall.x);
    clearPixel(playerBall.y+1,playerBall.x+1);
    clearPixel(playerBall.y+1,playerBall.x+2);
    clearPixel(playerBall.y+1,playerBall.x+3);
    clearPixel(playerBall.y+2,playerBall.x);
    clearPixel(playerBall.y+2,playerBall.x+1);
    clearPixel(playerBall.y+2,playerBall.x+2);
    clearPixel(playerBall.y+2,playerBall.x+3);
    clearPixel(playerBall.y+3,playerBall.x+1);
    clearPixel(playerBall.y+3,playerBall.x+2);
}

void GameScreen::createAllPlatforms()
{
    for (int n = 0; n < numPlatforms_ ; n++)
    {
        //create new platform and add to allPlatform array.
        Platform *newPlatform = new Platform;
        newPlatform->id = n;
        newPlatform->x = rand() % (maxX_-platGapSize_ + 1) - 1;
//randomlly select platform location
        newPlatform->y = maxY_ - n*platSpacing_ - platThickness_ + 1;
        allPlatforms[n] = newPlatform;
    }
}

//Garbage cleanup, free all the platforms!
void GameScreen::freeAllPlatforms()
{
    for (int n = 0; n < numPlatforms_ ; n++)
    {
        delete allPlatforms[n];
    }
}

void GameScreen::drawAllPlatforms()
```

```
{
    for (int n = 0; n < numPlatforms_ ; n++)
    {
        drawPlatform (allPlatforms[n]->x,allPlatforms[n]->y);
    }
}

void GameScreen::eraseAllPlatforms()
{
    for (int n = 0; n < numPlatforms_ ; n++)
    {
        erasePlatform (allPlatforms[n]->y);
    }
}

void GameScreen::shiftAllPlatforms(int n )
{
    for (int i = 0; i < numPlatforms_ ; i++)
    {
        if (allPlatforms[i]->y > (platThickness_+5+n))
            allPlatforms[i]->y = allPlatforms[i]->y - n;
        else
        {
            allPlatforms[i]->y =  maxY_ - platThickness_ + 1+6; //send back
to bottom.
            allPlatforms[i]->x = rand() % (maxX_-platGapSize_ + 1) - 1;
//select a new random position of gap!

        }
    }
}

Platform GameScreen::nextClosestPlatform(int y)
{
    int closestY = 100;
    Platform closestPlatform;
    //scan each platform and determin which is closest.
    for (int i = 0; i < numPlatforms_; i++)
    {

        if (((allPlatforms[i]->y - y) < closestY) && ((allPlatforms[i]->y -
y) >  -1))
        {
            closestY = allPlatforms[i]->y-y;
            closestPlatform = *allPlatforms[i];
        }
    }
    return closestPlatform;
}

//@Override of base class function
//had to override and rewrite since N5110 is written to write horizontal this
needs vertical
// this function reads existing font5x7 and rotate matrix to become font7x5
matrix.
//
// for example the number 4 font5x7 = 0x18,0x14,0x12,0x7F,0x10
```

```
//                            font7x5 = 0x02,0x06,0x10,0x12,0x1E,0x2,0x2
//then it is printed!
void GameScreen::printString(const char * str,int x,int y,bool invert)
{
    //set all pixels in rows to black
    if (invert){
        for (int a = 0; a < 48; a ++)
        {
            for (int b = y; b < (y+9); b++)
            {
                setPixel(b,a);
            }
        }
    }

    //scan through string, rotating font5x7 array and printing charector.
    while (*str)
    {

        //each byte in row
        for (int i = 0; i < 5; i++)
        {
            //each bit in byte
            for (int b = 0; b < 7; b++)
            {
                if (invert)
                    setPixel(y+b+1,x-i+1);

                if (font5x7[(*str - 32)*5 + i] & (1<<b)) //bitwise comparison
to mask at desired pixel
                {
                    setPixel(y+b+1,x-i+1);
                    if (invert)
                        clearPixel(y+b+1,x-i+1);
                }
                else
                {
                    if (!invert)
                        clearPixel(y+b+1,x-i+1);
                }
            }
        }
        str ++; //next char in string
        x -= 6;
    }
}

void GameScreen::displayStartScreen()
{
    clear();
    printString("LET",maxX_ - 14,-1);
    printString("THE BALL",maxX_ - 2,7);
    printString("DROP!",maxX_ - 12,16);
    printString("This way",maxX_ - 2,30);
    printString("------->",maxX_ - 2,37);
    printString("TO PLAY!",maxX_ - 2,45);
    printString("by: ",maxX_ - 2,61);
```

```
    printString("Thomas",maxX_ - 9,69);
    printString("Davies",maxX_ - 9,76);
    refresh();
}

void GameScreen::displayInstructionScreen()
{
    clear();
    printString("Hi Ball,",maxX_ - 2,-1);
    printString("You are",maxX_ - 2,7);
    printString("TRAPPED!",maxX_ - 2,15);
    printString("Hit the",maxX_ - 2,23);
    printString("roof and",maxX_ - 2,32);
    printString("you'll",maxX_ - 2,39);
    printString("surely",maxX_ - 2,47);
    printString("DIE!!",maxX_ - 2,55);
    printString("------->",maxX_ - 2,75);
    refresh();
}

void GameScreen::displayInstructionScreen2()
{
    clear();
    printString("CONTROLS",maxX_ - 2,-1);
    printString("Use the",maxX_ - 2,15);
    printString("joystick",maxX_ - 2,23);
    printString("to move",maxX_ - 2,32);
    printString("left or",maxX_ - 2,39);
    printString("right.",maxX_ - 2,47);
    printString("Click to",maxX_ - 2,55);
    printString("Pause!",maxX_ - 2,63);
    printString("------->",maxX_ - 2,75);
    refresh();
}

void GameScreen::displayCountdown()
{
    clear();
    Timer countdown;        //timer for countdown
    char buffer[10];
    countdown.start();
    printString("Survive!",maxX_ - 2,10);
    refresh();

    int prevTime = -1;
    //countdown from 5 to 1, in seconds.
    while (countdown.read() < 5)
    {
        if (floor(countdown.read()) > prevTime)
        {
            sprintf(buffer,"..%1.0f..",(5 - floor(countdown.read())));
            printString(buffer,maxX_ - 10,20 + 10*countdown.read());
            refresh();
            prevTime = countdown.read();
        }
    }
    countdown.stop();
```

```
}

void GameScreen::displayEndScreen(int lvl,int *cursor, bool isRecord)
{
	clear();
	char buffer [10];
	sprintf(buffer,"lvl: %d",lvl);
	printString("GameOver",maxX_ - 2,-1);
	printString("   :(  ",maxX_ - 2,7);
	printString(buffer,maxX_ - 2,23);
	printString("Reached!",maxX_ - 2,32);

	//only display record option if player achieved record!
	if (isRecord)
		printString("Record",maxX_ - 8,55,cursor[0]);

	printString("View",maxX_ - 15,63,cursor[1]);
	printString("Restart",maxX_ - 5,71,cursor[2]);
	refresh();
}

void GameScreen::displayRecordScreen(int *cursor,char ltr1,char ltr2,char
ltr3)
{
	clear();
	printString("Enter",maxX_ - 2,-1);
	printString("Initials",maxX_ - 2,7);

	//letter entry
	char buffer[2];
	sprintf(buffer,"%c",ltr1);
	printString(buffer,maxX_ - 23,20,cursor[0]);
	sprintf(buffer,"%c",ltr2);
	printString(buffer,maxX_ - 23,32,cursor[1]);
	sprintf(buffer,"%c",ltr3);
	printString(buffer,maxX_ - 23,44,cursor[2]);

	printString("Click",maxX_ - 2,59);
	printString("when",maxX_ - 2,67);
	printString("finished",maxX_ - 2,75);
	refresh();

}

void GameScreen::displayHighScores(char *s_hs0,char *s_hs1,char *s_hs2,int
*d_hs)
{
	clear();
	printString("SCORES",maxX_ - 2,-1);

	char buffer[10];

	sprintf(buffer,"%s %d",s_hs0,d_hs[0]);
	printString(buffer,maxX_ - 2,15);

	sprintf(buffer,"%s %d",s_hs1,d_hs[1]);
	printString(buffer,maxX_ - 2,33);
```

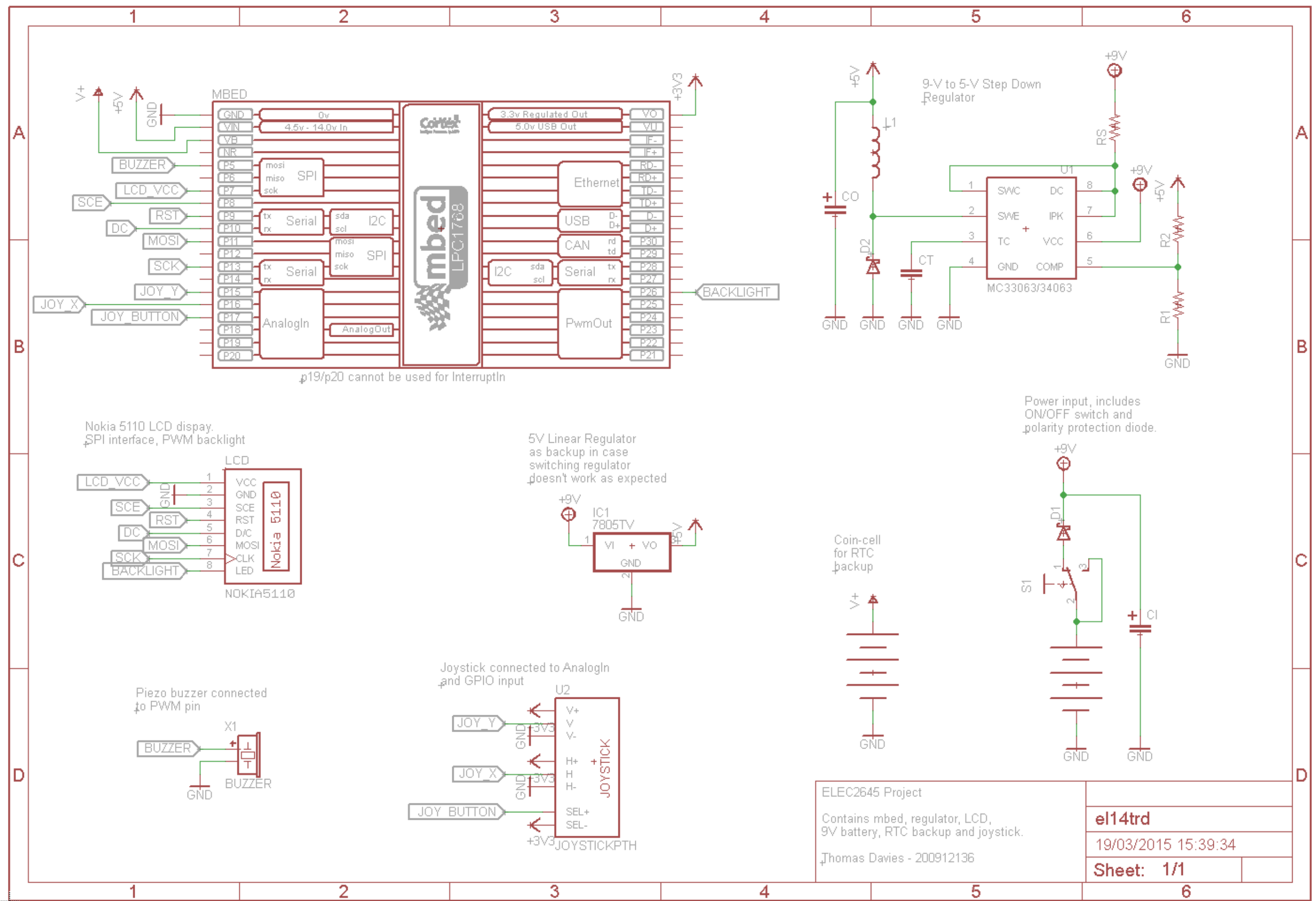
```
    sprintf(buffer,"%s %d",s_hs2,d_hs[2]);
    printString(buffer,maxX_ - 2,51);

    printString("Click to",maxX_ - 2,68);
    printString("reset!",maxX_ - 2,76);
    refresh();
}

void GameScreen::displayPauseScreen(int *cursor,char isSound)
{
    clear();
    printString("PAUSED",maxX_ - 2,7);

    char buffer[10];
    sprintf(buffer,"Sound? %c",isSound);
    printString(buffer,maxX_ - 2,30,cursor[0]);
    printString("Restart",maxX_ - 2,45,cursor[1]);
    printString("Return",maxX_ - 2,61,cursor[2]);
    refresh();
}
```

9.2 Schematic



9.3 PCB Design