DEPARTMENT OF COMPUTER SCIENCE AND APPLICATIONS
SHILLONG COLLEGE

CERTIFIED THAT THIS IS A BONAFIDE RECORD
OF THE PROJECT
ENTITLED

ABC is FUN

by

BANLAMKUPAR NONGKHLAW

Roll No: P1400004
Registration No: 14520 of 2013-14

GUIDE	*HEAD OF DEPARTMENT*	*EXAMINER*
Miss Ibameda Kharmawphlang	*Miss Aiom Mitri*	

CERTIFICATE

This is to certify that the Project work titled ' **ABC is FUN'** is a bonafide work done by Banlamkupar Nongkhlaw, RollNo: P1400004, Registration No: 14520 of 2013-14 under my guidance during the final year of the course.

Miss Ibameda Kharmawphlang

Internal Guide

Project seminar was held on.....at Shillong College, Shillong

Miss. Aiom MItri

Head of the Department

External Examiner

Table of content

ACKNOWLEDGEMENT

I would like to express a deep sense of gratitude and deep regards to my guide Miss Ibameda Kharmawphlang for his exemplary guidance, monitoring and encouragement throughout the thesis. And also I would like to show my respect to our HOD Miss AIOM MITRI and other staff for given me an opportunity and encourage me to do this project.

I am highly in debt to SHILLONG COLLEGE for giving this opportunity in fulfilment of my BCA degree course. Further some I am obliged to staff members of the COMPUTER APPLICATION DEPARTMENT for their crucial role and the valuable information provided by them in their respected fields. I am grateful for giving me the permission to use all the facilities and necessary materials with outmost cooperation during the period of my endeavor.

I feel thankful to all the personages for their guidance, encouragement and feedbacks from time without which this project would not be possible.

Above all I bow my head before the God Almighty whose blessings empowered me to complete this work successfully.

Introduction

Background of this project

Background is a set of events invented for a plot, presented as preceding and leading up to that plot. It is a literary device of a narrative history all chronologically earlier than the narrative of primary interest. In my project it's a single player game emphasizing logical thinking and planning. They often stress resources and time management, which usually takes precedence over fast action and character involvement. Tactical organization and execution are necessary, and the game creators usually place the decision-making skills and delivery of commands in the player's hands.

About the project

Its's a complete phonics game with different levels . The main character of " ABC is fun " is about a robot boy who go on collecting alphabets letters , but it is not easy to collect it , since there will be few traps lay waiting for him , if he can escape those traps then he can successfully collect another alphabets and go on collecting up till the end of alphabetic wise . There will be two levels in which the gamer/child can play . The main mission of the gamer is to collect alphabets so that he can identify the alphabetic letters one by one with the help of sound attach to the alphabets and also the gamer has to use a little bit of logic to avoid traps that are waiting for him .

Scope of my Game

This Report describe all the requirements for the project . The purpose of this research is to provide kids with the best ways to make them learn alphabets with fun and enjoy while learning . "ABC is fun" is a single - player phonics game on the Android platform . It also encourages creativity . I demonstrate the action flow between inputs , script , display (output)

Software Requirements Specification of "ABC is fun"

It includes the specification of this documentation with general description , specific requirements , and analysis of models . It also includes changes management of this requirement specification in case of any change .

Purpose :

This Software Requirements Specification (SRS) part is intended to give a complete overview of my Project the game "ABC is fun" including the action flow and initial user interface . The SRS document details all features upon which we have currently decided with reference to the manner and importance of their implementation.

Documents Conventions

There is a clear distinction , however , between the use of the words "player/gamer" and "character" . The "player" is a human being interacting with the game in the real world , while the "character" is the in - game player avatar being manipulated .

Intended Audience and Reading Suggestion

The SRS document also give project managers a way to ensure the game's audience to our original vision . Although the document may be read from front to back for a complete understanding of the project itself , see overall for Description . For a detailed description of the game - play elements and their interaction with the character , read System Features . Readers interested in the game - play interface and navigation between different front - end menus should go through External Interface Requirements .

Scope of this Document

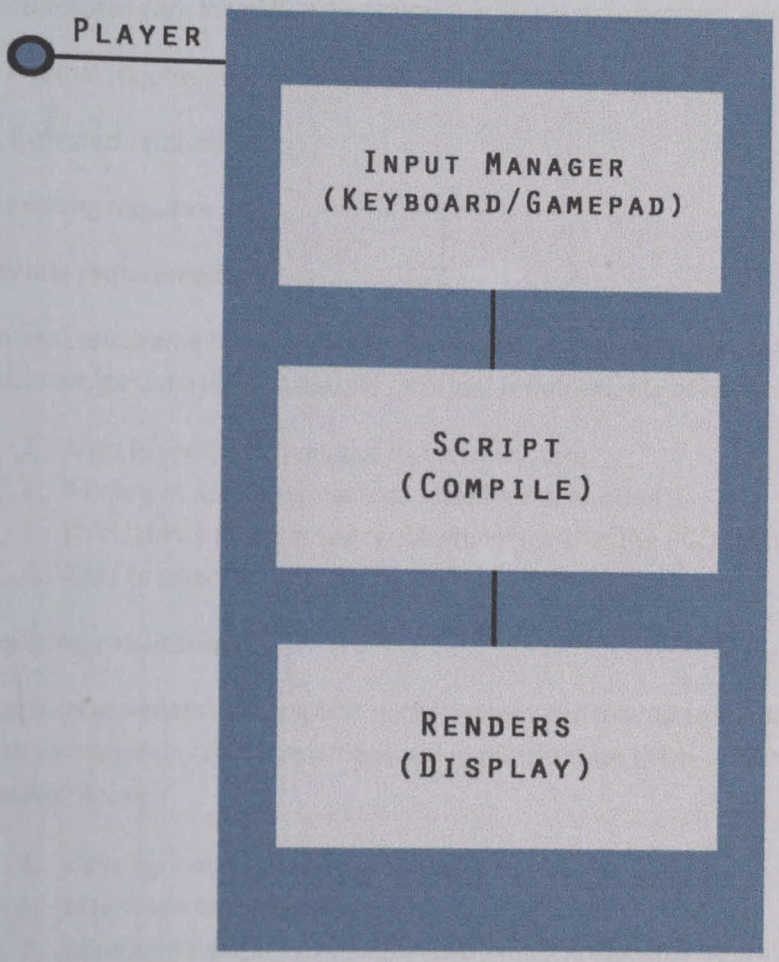
This Software Requirements Specification (SRS) describes the functional and nonfunctional requirements for the project . As I said before the purpose of this research is to provide a virtual image for the combination of both structured and unstructured information of my project "ABC is fun" .

The game will continue to grow until I feel it satisfactory for open – source distribution.

Product and Business Perspective of the Game

Software product development is a paradigm shift from routine application maintenance and support in the software industry . Developing a game/software product from scratch is a significant challenge for any individual if he has to do it alone for college project . It requires considerable investments in terms of cost and effort .

System Environment :



Gamer can interact with system by giving input (press key to start game) to the system . System give those inputs to script , if any change occur(if the value is changed) this object send to renders to display the things (a character can change its place).

Quality Function Deployment of "ABC is fun"

Quality Function Deployment ("QFD") is a technique that translates the need of the customer into technical requirements for software/game . It concentrates on maximizing customer satisfaction from the Software engineering process . With respect to my project the following requirements are identified by a QFD .

a. Normal requirements

b. Expected requirements

c. Exciting requirements

Normal requirements

Normal requirements consists of objectives and goals that are stated during the meeting with the actor/gamer/relevant people . Normal requirements of my project are :

1. User friendly efficient and lucrative system .
2. Minimum maintenance cost (Graphics defination) .
3. Availiability of expected requirements within the PC/Mobile configuration .
4. Easy to operate .

Expected requirements

These requirements are implicit to the system and maybe so fundamental that the actor/gamer/relevant people does not explicitly state them . Their absence will be a cause for dissatisfaction .

1. Develop system with limited cost .
2. Maximum high definition.
3. Minimum hardware requirements which is relevant for this game.
4. Design whole system with efficient manner.

Exiciting requirements

These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present:

1. Maximum high regulation with minimum hardware .
2. Easy to update .

Specific Requirements

This section covers the project external requirements of my Game and also indicates the user characteristics of this project.

External Interface requirements of the Game

User Interfaces

Every game must have a menu so it can be user friendly enough and gamers can easily fulfill their need . Menu is also an important thing while creating the SRS document section . In this SRS document part ; we have used the menu snapshots from the user manual part . These snapshots are based on the menu of the game .

Hardware Interfaces

"ABC is fun " is a mobile gaming application designed specifically for the android platform and is functional on both SMARTPHONES and TABLETS . Gaming application data is stored locally on the game engine elements . "ABC is fun" has been developed for Android Developed Version and all subsequent releases . Now the Android platform is graphically adaptable with a 2 Dimensional graphics library and a 3 Dimensional graphics library based on OpenGL ES 2.0 specification as well as hardware orientation , scaling , pixel format conversion and accelerated 3 Dimensions graphics .

Software Interface

"ABC is fun" has been developed using a series of game development tools .

Working tools and platform

1. Unity 3D (Version 5.2.1f1(64-bit))
2. Adobe Photoshop CS6(64-bit)
3. Android Software Development kit(Android SDK) : Software development kit for applications on the Android platform .
4. Adobe Audition.

User Characteristics for the System

There is only one user at a time in this software and the user interacts with the game (system) in a different manner .

So , Gamer/Player is the only one who communicates with the system through playing the game . And this Gamer/Player can be any person . The primary requirements is that , the Gamer/Player must read the playing procedure provided by me (The developer) .

Analysis Model of my Game project

This section describes the Software Requirements Specification of our project by analyzing the proper models of requirement engineering .

Scenario Based Model

This Model depicts how the user interacts with the system and the specific sequence of activities that occur as the software is used .

Use Case Scenario

The following table summarizes the cases of the system . I have created the use cases based on the UX view of the game . The swimlane diagram connects UX with background programming which are the two important views of a game SRS .

Use case diagram for use case Description

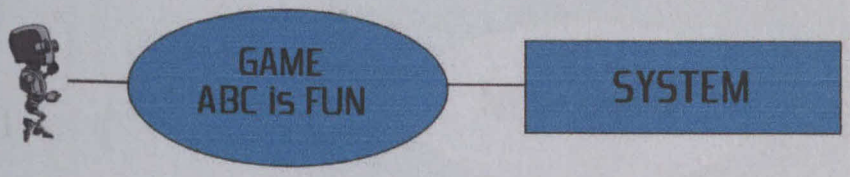


Fig 1 : Start level for Game

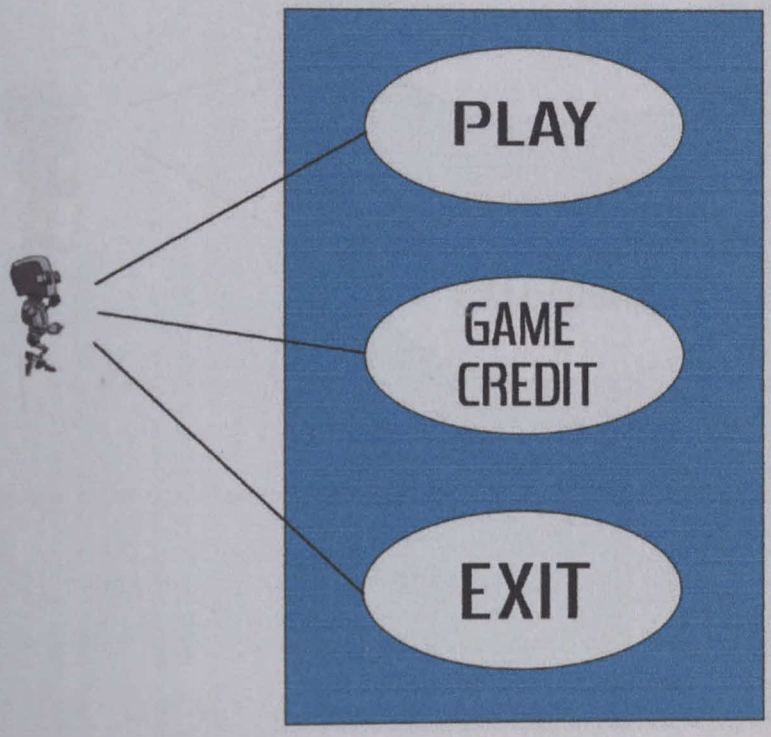


Fig 2: Level 1 for game UX

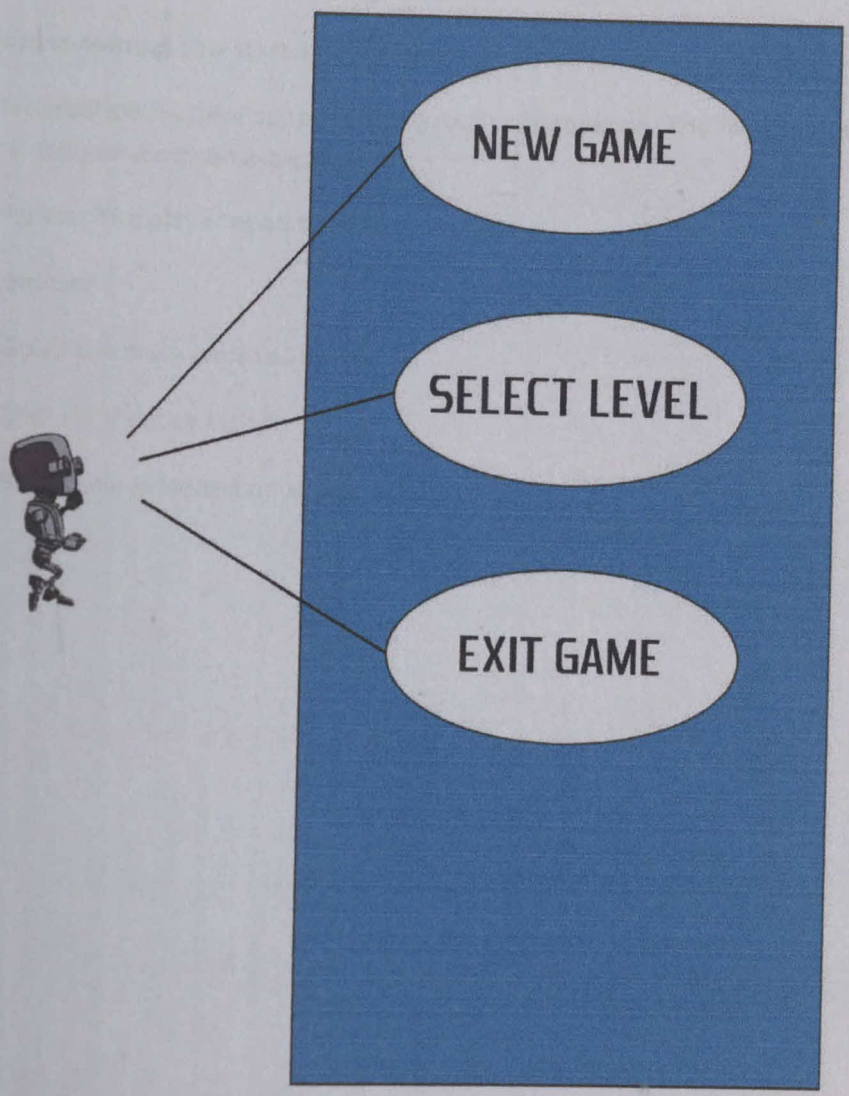


Fig 3 : Level 2 for game UX

This Diagram of Level 2.1(Fig 3) leads us to the "play" module of the use cases .

These use case description are given here –

Use case : New Game

Primary Actors : Anyone playing the game

Goal in context : To start a new game

Precondition : System supports the game configuration .The file has been triggered to run and the game screen appeared .

Triggers: The player need to start a new game

Scanario:

a.Go to the main menu of the game

b.Click on the play button

c.New game is loaded on the system

Use case: Select Level

Primary Actors: Anyone playing the game

Goal in context: To load the game from a required level

Precondition:

Game supports loading levels

Scenario:

a.Go to the main menu .

b.Click the select level button

c.Select a level

d.The level is loaded for play

Use case: Exit Game

Primary Actors: Anyone playing the game

Goal in context: To exit from the game level

Precondition: A game level is being played

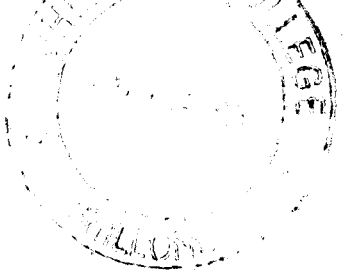
Triggers: Player needs to exit from the game level

Scenario:

a.Press game pause

b.When Pause Menu appears , click Return to Menu button

c.Game is exited and Title screen appears



Story

Use case:

Primary Actors: Anyone playing the game

Goal in context: It's a pheonic game , it's main goal is to make the children/player/gamer to recognanize letters .

Precondition:

a.Game is meant for kids only

Trigger: player wants to learn Alphabetics Letters

Scenario:

a.Go to main menu

b.Click play button

c.Phoenic game is being played

Quit

Use case: Quit

Primary Actors: Anyone playing the game

Goal in context: To Exit from the Game Process

Precondition: Player has entered in the game process

Triggers: Player needs to exit from the game

Scenario:

a.Go to the main menu

b.Click Quit button

c.Game is exited

Activity Diagram :

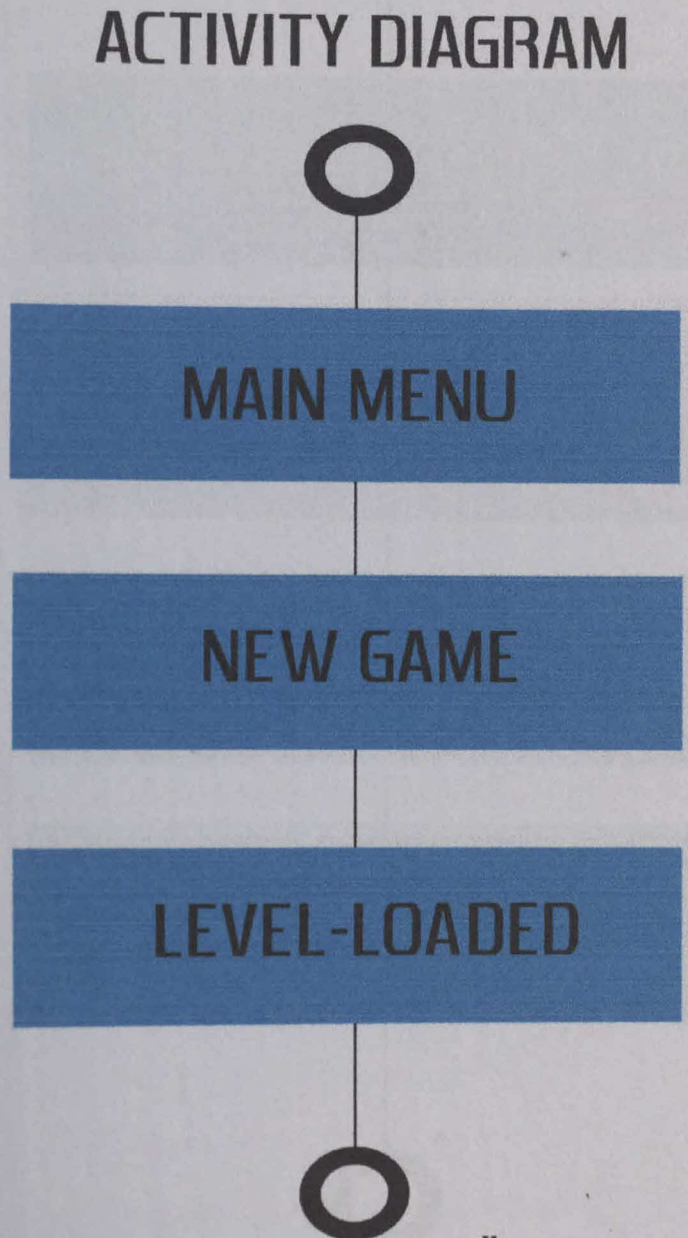


Fig 4 :Activity diagram for"new game ".

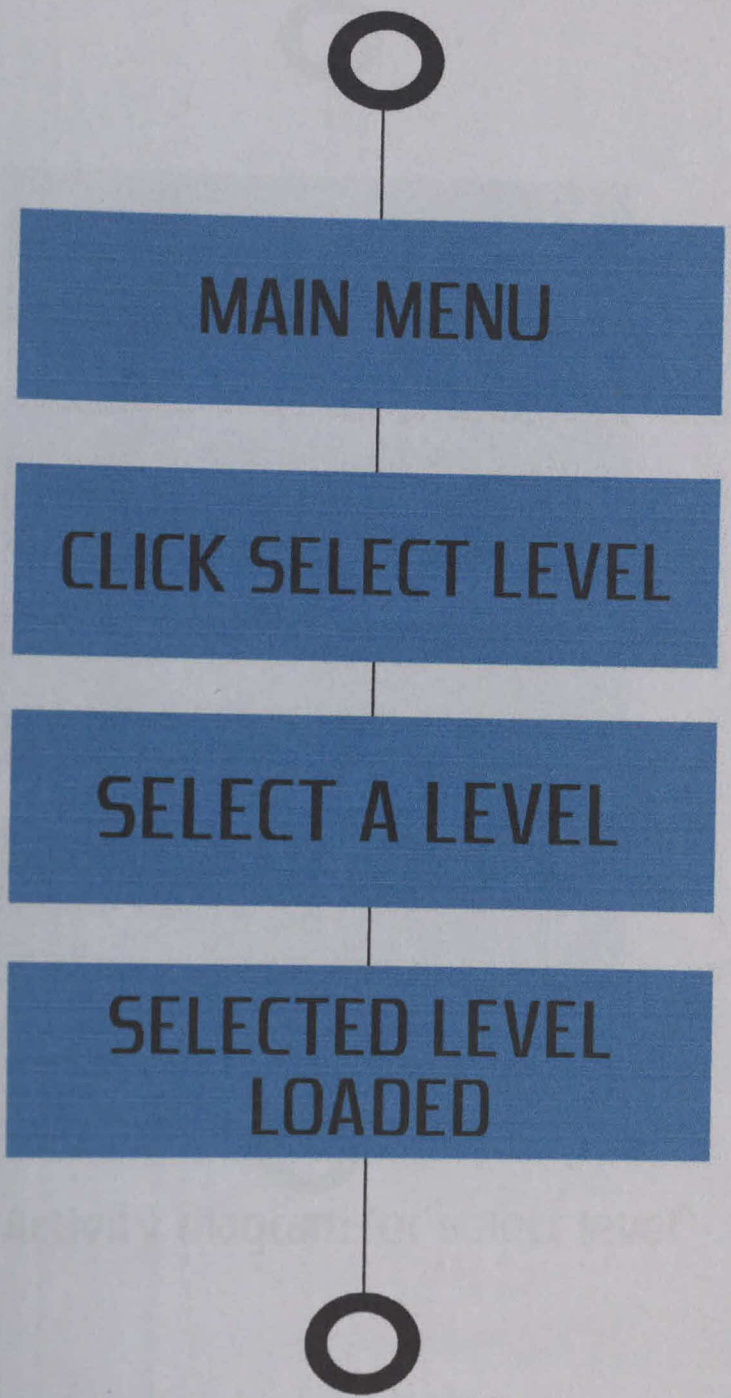


fig 5: Activity diagram for"select level"

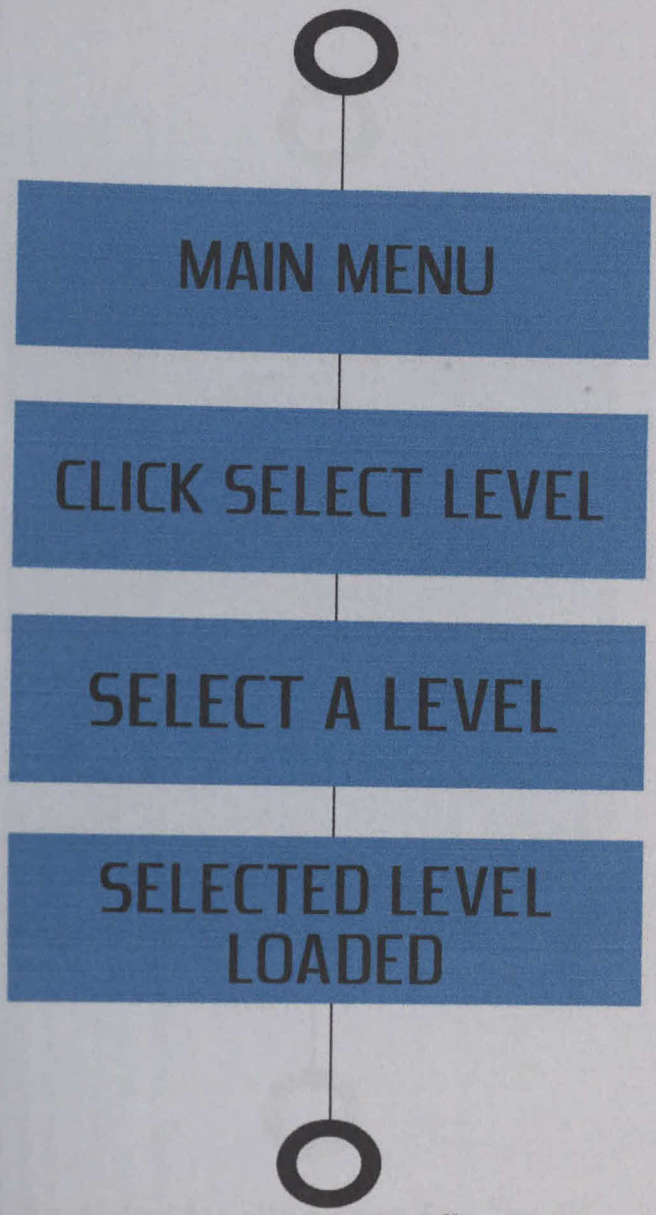


fig 5: Activity diagram for"select level"

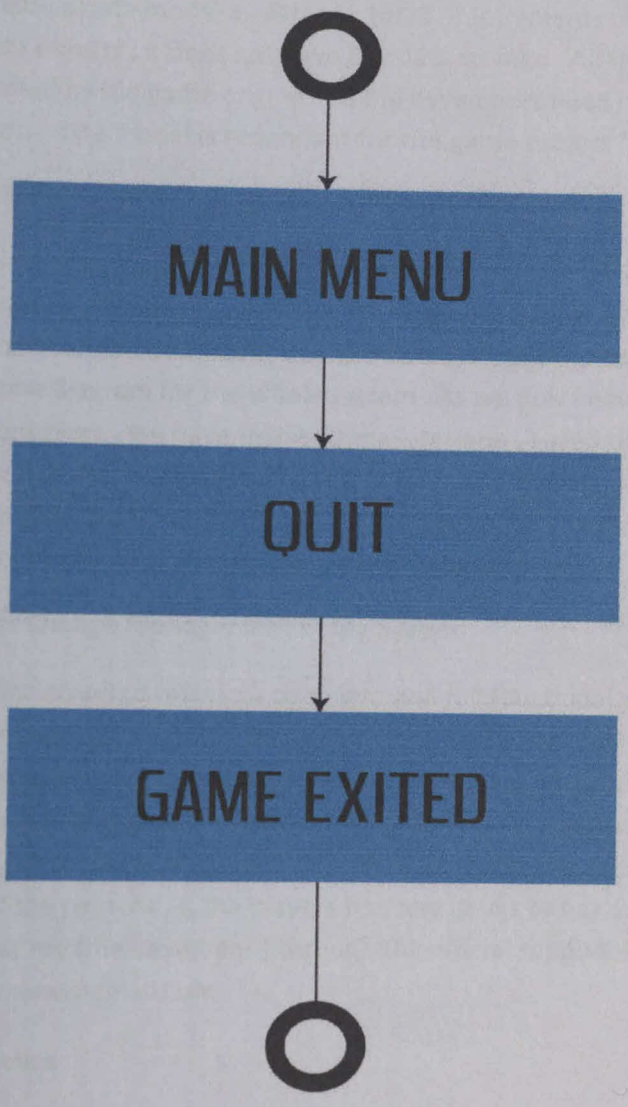


Fig 7 : Activity diagram for "quit"

Data Model

If software requirements include the need to create , extend or interface with database or if complex data structures must be constructed and manipulated , the software team must choose to create a data model as part of overall requirements modelling . Although my game has many data objects , it does not have any data storage . All the objects and their realated data are handled by the game engine . So the developers need not think about data storage . For this reason , data model is redundant for this game project .

Behavioral Model

The Behavioral indicates how software will respond to external events or stimuli . Thare are two ways to show these responses . One is state diagram and the other is sequence . Usually state diagram can be made in two ways , one is creating state diagram for each class and the other is to create a state diagram for the whole system . As we don't have any class , for this is not an object oriented game , we have followed the later one . I used the modules of the use case scenario to create the state diagram . And to lessen complexity I have divided the state diagram into two diagrams . On the other hand , for the sequence diagram , I have created separate a sequence diagram for all the used cases when necessary .

Requirements Change Management of my System

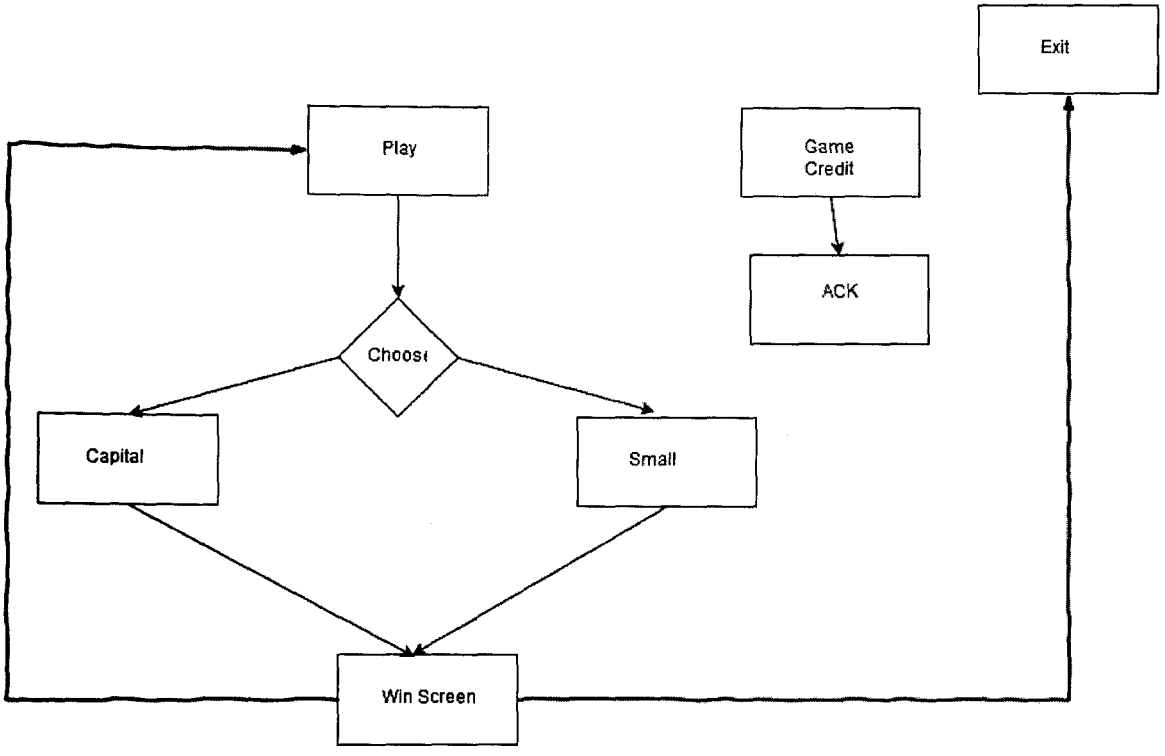
The developers intend to release a complete and full functional game that follows the guidelines mentioned in the SRS document . However , since the product will be released for multiple Android platforms(i.e., the different phones running the android system) , updates will likely be required . These updates would consists of any bug fixes that is necessary , compatibility patches for all the current phones that support the Android System , and expansions of the content . If the players find any issues or has any comments they would be able to contact me (the developer) through the official support email address which is banlamnongkhlaw@gmail.com

Bugs and Glitches

The players would be able to contact me through the support email system . This is where they would present any bugs or glitches they have detected and if they have any briefs that the game is not functioning properly . General concerns or comments would also need to be submitted here .

Design and Implimentation of "ABC is Fun"

Flow chart:



Product Design Terms

For every enterprise product two key terms of design is very important . They are :

a.Ux(User Experience)

b.Backend Programming

User Experience(UX)

To avoid unnecessary product features , simplifying design documentation and customer-facing technical public at , Incorporating business and marketing goals UX design is must.

User experience design(UXD or UED) is any aspects of a user's experience with a given system , including the interface , graphics , industrial design , physical interaction , and the manual in most cases ,

User experience Design fully encompasses traditional Human-Computer Interaction (HCI) design , and extends it by addressing all aspects of a product or service as perceived by the users . UX stands for mainly relevant access of usability , accessibility and HCI .

UX defines user experience as "a person's perceptions and responses that result from the use or anticipated use of a product , system or service" .

Backend Programming

The "backend" is the code supporting that front end(responsible for database access , business logic etc) .

In simple term , application front end is what you see(i.e. the user interface) and application backend is the application engine that you do not see . The "backend" is the code supporting that front end(responsible for database access , business logic etc) .

For efficient implementation , to increase user acceptance both two are very important in software industry .

Construction of the Game

Unity includes many build-in components which will expedite the process of game development immensely . These include :

Physics Engine

Collision Detection and Handling

Input Recognition

Object Creation and transformation Manipulation (position and rotation of game objects)

Scene integration (transition of one level to the next)

Model Attachment (representing game objects with 2D models)

Integration with Android

Unity3D's build settings simplify the process of transferring my game to the Android mobile device . After completing the project , or during any intermediary step for testing , we can select from the list of options , build the project , and upload it to one of our own devices . A separate license is required for this functionality .

Playing Procedure

Gamer first interact system UI to start playing . There are different levels in my game . Gamer can play each level in his own wish . The gamer can move the character so that he can collect coins and also at the same time learn alphabetic letters at the same time . After the player has finish a particular level he can move on to the next level so that he can learn some more.

Source code:

Camera 2D follow script:

```
using UnityEngine;

namespace UnitySampleAssets._2D
{

  public class Camera2DFollow : MonoBehaviour
  {

    public Transform target;
    public float damping = 1;
    public float lookAheadFactor = 3;
    public float lookAheadReturnSpeed = 0.5f;
```

```
    public float lookAheadMoveThreshold = 0.1f;
            public float yPosRestriction = -1;

    private float offsetZ;
    private Vector3 lastTargetPosition;
    private Vector3 currentVelocity;
    private Vector3 lookAheadPos;

    // Use this for initialization
    private void Start()
    {
      lastTargetPosition = target.position;
      offsetZ = (transform.position - target.position).z;
      transform.parent = null;
    }

    // Update is called once per frame
    private void Update()
    {

      // only update lookahead pos if accelerating or changed direction
      float xMoveDelta = (target.position - lastTargetPosition).x;

      bool updateLookAheadTarget = Mathf.Abs(xMoveDelta) > lookAheadMoveThreshold;

      if (updateLookAheadTarget)
      {
        lookAheadPos = lookAheadFactor*Vector3.right*Mathf.Sign(xMoveDelta);
      }
      else
      {
        lookAheadPos = Vector3.MoveTowards(lookAheadPos, Vector3.zero,
Time.deltaTime*lookAheadReturnSpeed);
      }

      Vector3 aheadTargetPos = target.position + lookAheadPos + Vector3.forward*offsetZ;
      Vector3 newPos = Vector3.SmoothDamp(transform.position, aheadTargetPos, ref
currentVelocity, damping);
```

```
                    newPos = new Vector3 (newPos.x, Mathf.Clamp (newPos.y,
yPosRestriction, Mathf.Infinity), newPos.z);
        transform.position = newPos;

        lastTargetPosition = target.position;
    }
  }
}
```

Camera follow script :

```
using UnityEngine;


namespace UnitySampleAssets._2D
{

  public class CameraFollow : MonoBehaviour
  {


    public float xMargin = 1f; // Distance in the x axis the player can move before the camera
follows.

    public float yMargin = 1f; // Distance in the y axis the player can move before the camera
follows.

    public float xSmooth = 8f; // How smoothly the camera catches up with it's target
movement in the x axis.

    public float ySmooth = 8f; // How smoothly the camera catches up with it's target
movement in the y axis.

    public Vector2 maxXAndY; // The maximum x and y coordinates the camera can have.

    public Vector2 minXAndY; // The minimum x and y coordinates the camera can have.


    private Transform player; // Reference to the player's transform.
```

```
    private void Awake()
    {
        // Setting up the reference.
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }
    private bool CheckXMargin()
    {
        // Returns true if the distance between the camera and the player in the x axis is greater
than the x margin.
        return Mathf.Abs(transform.position.x - player.position.x) > xMargin;
    }



    private bool CheckYMargin()

    {

        // Returns true if the distance between the camera and the player in the y axis is greater
than the y margin.

        return Mathf.Abs(transform.position.y - player.position.y) > yMargin;

    }

    private void Update()

    {

        TrackPlayer();

    }

 private void TrackPlayer()

    {

        // By default the target x and y coordinates of the camera are it's current x and y
coordinates.

        float targetX = transform.position.x;
```

```
        float targetY = transform.position.y;

        // If the player has moved beyond the x margin...
        if (CheckXMargin())
            // ... the target x coordinate should be a Lerp between the camera's current x position
and the player's current x position.
            targetX = Mathf.Lerp(transform.position.x, player.position.x,
xSmooth*Time.deltaTime);

        // If the player has moved beyond the y margin...
        if (CheckYMargin())
            // ... the target y coordinate should be a Lerp between the camera's current y position
and the player's current y position.
            targetY = Mathf.Lerp(transform.position.y, player.position.y,
ySmooth*Time.deltaTime);

        // The target x and y coordinates should not be larger than the maximum or smaller than
the minimum.
        targetX = Mathf.Clamp(targetX, minXAndY.x, maxXAndY.x);
        targetY = Mathf.Clamp(targetY, minXAndY.y, maxXAndY.y);

        // Set the camera's position to the target position with the same z component.
        transform.position = new Vector3(targetX, targetY, transform.position.z);
    }
  }
}
```

Collectable alphabet script :

```
using UnityEngine;
using System.Collections;

public class CollectableAlphabet : MonoBehaviour {
                                public LevelManager levelManager;
                                 //public AudioClip pickupSound;
                                    // Use this for initialization
                                          void Start () {
                                                  //levelManager =
GameObject.FindObjectOfType<LevelManager>();
                                                 }

                                 // Update is called once per frame
                                          void Update () {

                                                 }
                               void OnTriggerEnter2D(Collider2D target){
                                                  if (gameObject.tag == "Player") {
                                                        //if(pickupSound){

                    //AudioSource.PlayClipAtPoint(pickupSound,transform.position);
                                                                    //Destroy(gameObject);
                                                        //}
                                                        levelManager.LoadLevel
("WinStage");
                                                        //Destroy (gameObject);
                                                  }
                                                  Destroy (gameObject);
                                                  //SimulateWin();
                                                 }
                                        //void SimulateWin(){
                                                  //levelManager.LoadLevel ("WinStage");
                                               //}

}
```

Collectable Sounds script:

```
using UnityEngine;
using System.Collections;

public class CollectableSounds : MonoBehaviour {
                                    public AudioClip pickupSound;
                                      // Use this for initialization
                                            void Start () {

                                            }

                                    // Update is called once per frame
                                            void Update () {

                                            }
                                void OnTriggerEnter2D(Collider2D target)
                                            {
                                                if (target.gameObject.tag == "Player") {
                                                        if(pickupSound)

                        AudioSource.PlayClipAtPoint(pickupSound,transform.position);
                                                        Destroy(gameObject);
                                                }
                                            }
}
```

Level manager script :

```
using UnityEngine;
using System.Collections;

public class LevelManager : MonoBehaviour {

                                    // Use this for initialization
                                public void LoadLevel(string name){
                                                Debug.Log ("New Level load: " + name);
                                                Application.LoadLevel (name);
                                            }
                                    public void QuitRequest () {
                                                Debug.Log ("Quit requested");
                                                Application.Quit ();
```

}
}

Parallaxing script :

```
using UnityEngine;
using System.Collections;

public class Parallaxing : MonoBehaviour {

    public Transform[] backgrounds;                    // Array (list) of all the back- and
foregrounds to be parallaxed
    private float[] parallaxScales;                    // The proportion of the camera's
movement to move the backgrounds by
    public float smoothing = 1f;                       // How smooth the parallax is going
to be. Make sure to set this above 0

    private Transform cam;                                          // reference to the
main cameras transform
    private Vector3 previousCamPos;                    // the position of the camera in the
previous frame

    // Is called before Start(). Great for references.
    void Awake () {
        // set up camera the reference
        cam = Camera.main.transform;
    }

    // Use this for initialization
    void Start () {
        // The previous frame had the current frame's camera position
        previousCamPos = cam.position;

        // asigning coresponding parallaxScales
        parallaxScales = new float[backgrounds.Length];
        for (int i = 0; i < backgrounds.Length; i++) {
                        parallaxScales[i] = backgrounds[i].position.z*-1;
        }
    }
```

```
    // Update is called once per frame
    void Update () {

        // for each background
        for (int i = 0; i < backgrounds.Length; i++) {
        // the parallax is the opposite of the camera movement because the previous frame
multiplied by the scale
            float parallax = (previousCamPos.x - cam.position.x) * parallaxScales[i];

            // set a target x position which is the current position plus the parallax
              float backgroundTargetPosX = backgrounds[i].position.x + parallax;

        // create a target position which is the background's current position with it's target x
position
            Vector3 backgroundTargetPos = new Vector3 (backgroundTargetPosX,
backgrounds[i].position.y, backgrounds[i].position.z);

              // fade between current position and the target position using lerp
        backgrounds[i].position = Vector3.Lerp (backgrounds[i].position, backgroundTargetPos,
smoothing * Time.deltaTime);
        }

        // set the previousCamPos to the camera's position at the end of the frame
        previousCamPos = cam.position;
    }
}
```

Pause Menu script :

```
using UnityEngine;
using System.Collections;

public class PauseMenu : MonoBehaviour {
    //public string levelSelect;
    public string mainMenu;
    public bool isPaused;
    public GameObject pauseMenuCanvas;

    // Update is called once per frame
    void Update ()
```

```
    {
        if (isPaused) {
                        pauseMenuCanvas.SetActive (true);
        } else
        {
                        pauseMenuCanvas.SetActive(false);
        }

        if (Input.GetKeyDown (KeyCode.Escape))
        {
                            isPaused = !isPaused;
        }
    }
    public void Resume()
    {
        isPaused = false;
    }
    /*public void levelSelect()
    {
        Application.LoadLevel (levelSelect);
    }*/
    public void Quit()
    {
        Application.LoadLevel (mainMenu);
    }
}
```

Platformer 2D user control script :

```
using UnityEngine;
using UnitySampleAssets.CrossPlatformInput;

namespace UnitySampleAssets._2D
{

  [RequireComponent(typeof (PlatformerCharacter2D))]
  public class Platformer2DUserControl : MonoBehaviour
  {
    private PlatformerCharacter2D character;
    private bool jump;
```

```
    private void Awake()
    {
        character = GetComponent<PlatformerCharacter2D>();
    }

    private void Update()
    {
        if(!jump)
        // Read the jump input in Update so button presses aren't missed.
        jump = CrossPlatformInputManager.GetButtonDown("Jump");
    }

    private void FixedUpdate()
    {
        // Read the inputs.
        bool crouch = Input.GetKey(KeyCode.LeftControl);
        float h = CrossPlatformInputManager.GetAxis("Horizontal");
        // Pass all parameters to the character control script.
        character.Move(h, crouch, jump);
        jump = false;
    }
  }
}
```

Platformer character 2D script :

```
using UnityEngine;

//namespace UnitySampleAssets._2D
//{

  public class PlatformerCharacter2D : MonoBehaviour
  {
    private bool facingRight = true; // For determining which way the player is currently facing.

    [SerializeField] private float maxSpeed = 10f; // The fastest the player can travel in the x
axis.
    [SerializeField] private float jumpForce = 400f; // Amount of force added when the player
jumps.
```

```
    [Range(0, 1)] [SerializeField] private float crouchSpeed = .36f;
                                // Amount of maxSpeed applied to crouching movement. 1 =
100%

    [SerializeField] private bool airControl = false; // Whether or not a player can steer while
jumping;
    [SerializeField] private LayerMask whatIsGround; // A mask determining what is ground to
the character

    private Transform groundCheck; // A position marking where to check if the player is
grounded.
    private float groundedRadius = .2f; // Radius of the overlap circle to determine if grounded
    private bool grounded = false; // Whether or not the player is grounded.
    private Transform ceilingCheck; // A position marking where to check for ceilings
    private float ceilingRadius = .01f; // Radius of the overlap circle to determine if the player
can stand up
    private Animator anim; // Reference to the player's animator component.


    private void Awake()
    {
      // Setting up references.
      groundCheck = transform.Find("GroundCheck");
      ceilingCheck = transform.Find("CeilingCheck");
      anim = GetComponent<Animator>();
    }


    private void FixedUpdate()
    {
      // The player is grounded if a circlecast to the groundcheck position hits anything
designated as ground
      grounded = Physics2D.OverlapCircle(groundCheck.position, groundedRadius,
whatIsGround);
      anim.SetBool("Ground", grounded);

      // Set the vertical animation
```

```
        anim.SetFloat("vSpeed", GetComponent<Rigidbody2D>().velocity.y);
    }


    public void Move(float move, bool crouch, bool jump)
    {


        // If crouching, check to see if the character can stand up
        if (!crouch && anim.GetBool("Crouch"))
        {
            // If the character has a ceiling preventing them from standing up, keep them
crouching
            if (Physics2D.OverlapCircle(ceilingCheck.position, ceilingRadius, whatIsGround))
                crouch = true;
        }

        // Set whether or not the character is crouching in the animator
        anim.SetBool("Crouch", crouch);

        //only control the player if grounded or airControl is turned on
        if (grounded || airControl)
        {
            // Reduce the speed if crouching by the crouchSpeed multiplier
            move = (crouch ? move*crouchSpeed : move);

            // The Speed animator parameter is set to the absolute value of the horizontal input.
            anim.SetFloat("Speed", Mathf.Abs(move));

            // Move the character
            GetComponent<Rigidbody2D>().velocity = new Vector2(move*maxSpeed,
GetComponent<Rigidbody2D>().velocity.y);

            // If the input is moving the player right and the player is facing left...
            if (move > 0 && !facingRight)
                // ... flip the player.
                Flip();
                // Otherwise if the input is moving the player left and the player is facing right...
```

```
        else if (move < 0 && facingRight)
            // ... flip the player.
            Flip();
        }
        // If the player should jump...
        if (grounded && jump && anim.GetBool("Ground"))
        {
            // Add a vertical force to the player.
            grounded = false;
            anim.SetBool("Ground", false);
            GetComponent<Rigidbody2D>().AddForce(new Vector2(0f, jumpForce));
        }
    }


    private void Flip()
    {
        // Switch the way the player is labelled as facing.
        facingRight = !facingRight;

        // Multiply the player's x local scale by -1.
        Vector3 theScale = transform.localScale;
        theScale.x *= -1;
        transform.localScale = theScale;
    }
  }
//}
```

Tiling script :

```
using UnityEngine;
using System.Collections;

[RequireComponent (typeof(SpriteRenderer))] // once you attach a script if you have't create
unity will create one

public class Tiling : MonoBehaviour {

    public int offsetX = 2;                                    // the offset so that we don't get any
weird errors between foreground and brackground when they collide to form a new platform
```

```
    // these are used for checking if we need to instantiate stuff
    public bool hasARightBuddy = false; // by default to false otherwise it will create errors
    public bool hasALeftBuddy = true;

    public bool reverseScale = false;// used if the object is not tilableSimilar to Sliced, but tiles
(repeats).For sprites with no borders at all, the entire sprite is tiled.

    private float spriteWidth = 0f;                    // the width of our element
    private Camera cam;
    private Transform myTransform;

    void Awake () {
        cam = Camera.main;
        myTransform = transform;
    }

    // Use this for initialization
    void Start () {
        SpriteRenderer sRenderer = GetComponent<SpriteRenderer>(); // only one sprite
renderer
        spriteWidth = sRenderer.sprite.bounds.size.x; // give us the width of our
element/sprite
    }

    // Update is called once per frame
    void Update () {
        // does it still need buddies? If not do nothing
        if (hasALeftBuddy == true || hasARightBuddy == false)
        {
        // calculate the cameras extend (half the width) of what the camera can see in world
coordinates
        float camHorizontalExtend = cam.orthographicSize * Screen.width/Screen.height; //
length from centre to the right camera

        // calculate the x position where the camera can see the edge of the sprite (element)
          float edgeVisiblePositionRight = (myTransform.position.x + spriteWidth/2) -
camHorizontalExtend;
```

```
        float edgeVisiblePositionLeft = (myTransform.position.x - spriteWidth/2) +
camHorizontalExtend;

        // checking if we can see the edge of the element and then calling MakeNewBuddy if
we can
        if (cam.transform.position.x >= edgeVisiblePositionRight - offsetX && hasARightBuddy
== false)
                                                {
                                                    MakeNewBuddy (1);
                                                    hasARightBuddy = true;
                                                }
            else if (cam.transform.position.x <= edgeVisiblePositionLeft + offsetX &&
hasALeftBuddy == true)
                                                {
                                                    MakeNewBuddy (-1);
                                                    hasALeftBuddy = false;
                                                }
        }
    }

    // a function that creates a buddy on the side required
    void MakeNewBuddy (int rightOrLeft) {
        // calculating the new position for our new buddy
        Vector3 newPosition = new Vector3 (myTransform.position.x + spriteWidth *
rightOrLeft, myTransform.position.y, myTransform.position.z);
        // instantating our new body and storing him in a variable
        Transform newBuddy = Instantiate (myTransform, newPosition, myTransform.rotation)
as Transform;

        // if not tilable let's reverse the x size og our object to get rid of ugly seams
        if (reverseScale == true) {
                newBuddy.localScale = new Vector3 (newBuddy.localScale.x*-1,
newBuddy.localScale.y, newBuddy.localScale.z);
        }

        newBuddy.parent = myTransform.parent;
        if (rightOrLeft > 0) {
                newBuddy.GetComponent<Tiling>().hasALeftBuddy = false;
```

```
        }
        else {
                newBuddy.GetComponent<Tiling>().hasARightBuddy = true;
        }
    }
}
```

Mobilecontrolrig script :

```
using UnityEngine;


namespace UnitySampleAssets.CrossPlatformInput
{
#if UNITY_EDITOR
  using UnityEditor;
  [ExecuteInEditMode]
#endif
  public class MobileControlRig : MonoBehaviour
  {

    // this script enables or disables the child objects of a control rig
    // depending on whether the USE_MOBILE_INPUT define is declared.

    // This define is set or unset by a menu item that is included with
    // the Cross Platform Input package.

#if !UNITY_EDITOR
   void OnEnable()
   {
       CheckEnableControlRig();
   }
   #endif

#if UNITY_EDITOR

    private void OnEnable()
    {
      EditorUserBuildSettings.activeBuildTargetChanged += Update;
      EditorApplication.update += Update;
```

```
    }

    private void OnDisable()
    {
      EditorUserBuildSettings.activeBuildTargetChanged -= Update;
      EditorApplication.update -= Update;
    }

    private void Update()
    {
      CheckEnableControlRig();

    }
#endif

    private void CheckEnableControlRig()
    {
#if MOBILE_INPUT
        EnableControlRig(true);
        #else
      EnableControlRig(false);
#endif

    }

    private void EnableControlRig(bool enabled)
    {
      foreach (Transform t in transform)
      {
        t.gameObject.SetActive(enabled);
      }
    }
  }
}
```

Joystick script:

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnitySampleAssets.CrossPlatformInput;
```

```
public class Joystick : MonoBehaviour , IPointerUpHandler , IPointerDownHandler ,
IDragHandler {

  public int MovementRange = 100;

  public enum AxisOption
  {                                  // Options for which axes to use
    Both,                                        // Use both
    OnlyHorizontal,                                  // Only horizontal
    OnlyVertical                                   // Only vertical
  }

  public AxisOption axesToUse = AxisOption.Both;  // The options for the axes that the still will
use
  public string horizontalAxisName = "Horizontal";// The name given to the horizontal axis for
the cross platform input
  public string verticalAxisName = "Vertical";   // The name given to the vertical axis for the
cross platform input

  private Vector3 startPos;
  private bool useX;                                 // Toggle for using the x axis
  private bool useY;                                 // Toggle for using the Y axis
  private CrossPlatformInputManager.VirtualAxis horizontalVirtualAxis;          // Reference to
the joystick in the cross platform input
  private CrossPlatformInputManager.VirtualAxis verticalVirtualAxis;          // Reference to
the joystick in the cross platform input

  void Start () //DCURRY
   {

    startPos = transform.position;
    CreateVirtualAxes ();
  }

  private void UpdateVirtualAxes (Vector3 value) {

    var delta = startPos - value;
```

```
        delta.y = -delta.y;
        delta /= MovementRange;
        if(useX)
        horizontalVirtualAxis.Update (-delta.x);

        if(useY)
        verticalVirtualAxis.Update (delta.y);

    }

    private void CreateVirtualAxes()
    {
        // set axes to use
        useX = (axesToUse == AxisOption.Both || axesToUse == AxisOption.OnlyHorizontal);
        useY = (axesToUse == AxisOption.Both || axesToUse == AxisOption.OnlyVertical);

        // create new axes based on axes to use
        if (useX)
            horizontalVirtualAxis = new
CrossPlatformInputManager.VirtualAxis(horizontalAxisName);
        if (useY)
            verticalVirtualAxis = new CrossPlatformInputManager.VirtualAxis(verticalAxisName);
    }


    public  void OnDrag(PointerEventData data) {

        Vector3 newPos = Vector3.zero;

        if (useX) {
            int delta = (int) (data.position.x - startPos.x);
            // delta = Mathf.Clamp(delta,  - MovementRange,  MovementRange); //DCURRY
            newPos.x = delta;
        }

        if (useY)
        {
            int delta = (int)(data.position.y - startPos.y);
```

```
        //delta = Mathf.Clamp(delta, -MovementRange,  MovementRange); //DCURRY
        newPos.y = delta;
    }
        transform.position = Vector3.ClampMagnitude(new Vector3(newPos.x ,newPos.y
,newPos.z),MovementRange)+startPos;
    UpdateVirtualAxes (transform.position);
  }


  public  void OnPointerUp(PointerEventData data)
  {
    transform.position = startPos;
    UpdateVirtualAxes (startPos);
  }


  public  void OnPointerDown (PointerEventData data) {
  }

  void OnDisable () {
    // remove the joysticks from the cross platform input
    if (useX)
    {
      horizontalVirtualAxis.Remove();
    }
    if (useY)
    {
      verticalVirtualAxis.Remove();
    }
  }
}
```

uiManager script :

```
using UnityEngine;
using System.Collections;

public class uiManager : MonoBehaviour {

  // Use this for initialization
```

```
    void Start ()
    {

    }

    // Update is called once per frame
    void Update ()
    {

    }
    public void Pause()
    {
        if(Time.timeScale == 1)
        {
                                    Time.timeScale = 0;
        }else if(Time.timeScale == 0)
        {
                                    Time.timeScale = 1;
        }
    }
}
```

Results

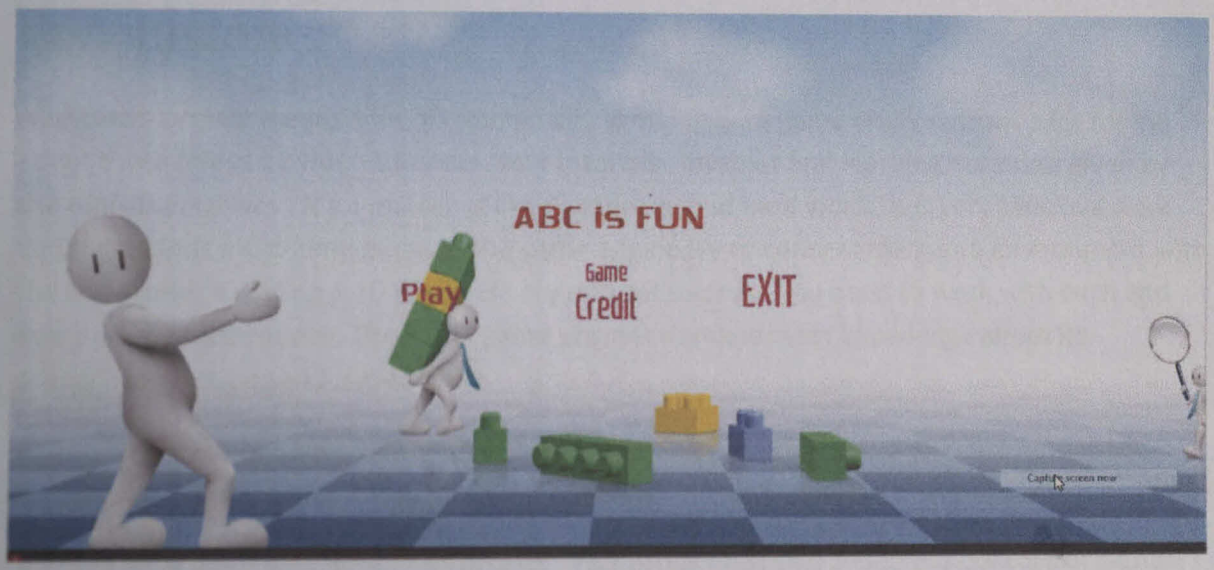


Fig : main menu

Fig : Play scene

Conclusion

A software project means a lot of experience . Working with game engine means a lot for me . I adopt these things by video tutorials , text tutorials , internet and learning materials given by the tools themselves . It's a matter of time , patience and hard work . It is very sensitive work and it demands much time because the game engine try to connect the game environment with the real world . Creating a 2D model is very difficult because you need to work with each and every point of the model . The Exists game engines demands vast knowledge about its properties , sections and sub-sections .

Now , I have know much more about game engines . How it works ? The properties , objects and others . I now know how a model is constructed and how it is animated .

I have learn a lot about this project . This project has sharpened our concept of game engine , animation and the software-hardware interface . The piece of software I developed is intended to serve for educational purpose only . The success of this project may give pleasure to gamers .

There were times that I almost lost hope but I recover through constant concentration and hard work .

References

1. http://www.mixamo/
2. http://the3dmodels.com/stuff,
3. http://www.unity3dstudent.com
4. http://students.autodesk.com/
5. http://digitaltutors.com