GAME 255 FINAL EXAM

# 5/10/2016

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Victor Haskins \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Short Answer

|  |
| --- |
| Answer each question fully. |
| Use full sentences. |

*What are the four major game design considerations for a Roguelike game?*

A roguelike should offer replayability due to random Levels and tile-based graphics.

A roguelike should be fundamentally simple to produce.

A roguelike should be built with accountability for all actions and buys a player makes in mind.

A roguelike should be challenging for the player.

*What are the 3 T’s to Mobile development (name and definition), and why are they important?*

The first is “Download Time”; it is the amount of time that the game will take to download onto the phone. Download Time is important because people are impatient and want to play the new games they choose quickly. If the game is too large or too complicated to download, then the Download Time will increase. If that time gets too large, then the client will cancel and not play the game.

The second is “Play Time”; this is the amount of time that a player should expect to play in a sitting, whether it will be a few minutes or several hours. Play Time is important because different game genres and types will have different play times that will keep it engaging. Most mobile games focus on having play times that are only a few minutes at a time for a full experience since they will try to squeeze in a play session in transit or when relieving oneself.

The final is Battery Time; this is the amount of time that the battery can hold out while playing your game. The more graphically and/or computationally intensive the game is, the more quickly a phone’s battery life will be eaten up. Furthermore, if trying to keep up with massive computation workloads for long enough, the battery will quickly start to heat up. Repeated overheating on the battery will start to shorten the battery life and eventually burn it out altogether. A developer needs to keep this in mind when making a game so the player doesn’t kill their phone or other mobile device when enjoying their title.

*What is the difference between random generation and procedural generation?*

Procedural generation takes Non-Existant Features that are then seeded into the level on-the-fly, controlled by designer, (and sometimes user,) inputs. Random generation takes Existant Features and populates a game level with them.

*Name two in-app monetization techniques, and when they are appropriate to use.*

The first technique would be to add the microtransaction option to purchase in-game currency in free-to-play mode if there is a viable system for such a currency that does not give the wealthier players an unfair advantage. An example of such would be to purchase an element that may increase the opportunities to play or speed up gameplay. The currency should be something that the average player can earn on their own over time.

The second option would be to include in-app ads can be shown between play sessions or before a play session, (with a prompt to get a boost if the ad is watched in its entirety.)

*When switching between different phones (with different* ***resolutions****) what do you need to do?*

Either have the game recognize the size, (and orientation sometimes,) of the screen and then either play a scene designed for that resolution or have the camera and elements of the scenes compensate for that OR have the elements of your scene squash or stretch according to the resolution. So it is still readable in the game.

*What must you do for every Android project before it can be built?*

Go to the Build Settings and change the format to Android. Then, as the project is building, you will be prompted for the android-sdk folder. Choose this folder from the provided window.

In the editor settings, set the input device from “none” to “any android device”. Set the company name and game name of the game, and then set the build name to “com.\**company name*\*.\**game name*\*”.

To test your game before the build, connect an android device to the computer, open the command window, find the “platform-tools” folder inside the Android folder and run the command “adb devices”.

*What is object pooling, and why is it so importan****t****?*

Have a set number of items created at the start of the game that you expect to instantiate and/or destroy with high frequency and disable all of them. When needing one or more instances, an instance that isn’t being used is set to the appropriate position, rotation, and other viable variables, then enable it. At the end of its individual runtime, it is instead disabled instead of destroyed. That makes the object a viable target for the next time an instance is needed.

*What is event messaging, and why do you not use the built-in Unity event messaging system?*

This is the operation of sending a prompt to the different events in the system to run given certain criteria. If they are populated, then the appropriate events are run. Event messaging is incredibly expensive, computationally, and will heat up the battery exceptionally fast. To keep the battery and battery life of the device healthy, avoid this option whenever possible.

*What is Box2D? How is it used?*

It is the premiere two dimensional physics engine, used to show collisions of a 2D game whether they have a 2D or 3D design space. The version that we were exposed to in class was written entirely in C++. Recreating or rewriting this engine is a surefire way to show your programming expertise.

*Is the 2D physics system or 3D physics system more resource-intense?*

The 3D system since there is another dimension to keep track of as well as up to 2 more rotation axes, (the notable exception being the Paper Mario franchise.)

*Is it safe to store information like purchases and high scores locally using File.IO on a mobile device?*

Absolutely not; data in a format like that can be hacked and modified and then you don’t see a penny. This sort of information needs to be stored on a database and referenced online every time the currency or purchase system is brought up.

*How many significant digits is acceptable for a floating point number in a mobile application? Why does it matter?*

Two significant digits are all that is acceptable on mobile because more than that starts becoming more costly for calculations and will start to create errors with loading, saving, rounding, etc.

*What is a singleton, and why are they especially important in mobile applications?*

A singleton is an object that uses a static reference of itself to make sure that there is only one copy of itself during gameplay. This can be used to ensure coding structure doesn’t have multiple instances of the same object in play that might be used for various purposes, such as a database of available objects, or an object used for logging and file I/O.

*What use is a ScriptableObject in mobile development? Include at least two different examples of how/why are they used. (Don’t give code)*

A scriptable object is a file that can itself be considered an object. This can then be instantiated in the game and saved as a prefab. From there, one can manipulate the meta files to either force the code to binary for a weak form of encoding or to text for easy developer manipulation.

*When using the accelerometer and gyroscope, there are two important things you should include to increase the usability. What are they and why do they matter?*

My answer was to add a centering option that reset the inputs for the game in case they become out of whack and need to start fresh with their input.

A setting should be added to adjust the sensitivity of the object’s accelerometer and gyroscope to accommodate for different people’s preferences or specific situations, (like cramped locations.)

There should also be a “dead zone” that won’t register minor movements away from zero vectors so minor twitches while playing with the phone zeroed out won’t cause the player to lose.

*Give an example of when the accelerometer would be a good choice to use on a device (you cannot use any in-class projects as this example – never is also not an acceptable answer):*

When wanting to include a pedometer system in your game, either as a minigame or as a limited currency accumulator, (like what the 3DS does.)

*Give an example of how to capture a single-touch input (pseudo-code or actual code):*

void Update () {

if(Input.touchCount == 1)

{

if(Input.GetTouch(0).phase == TouchPhase.Began)

{

v2Previous = Input.GetTouch(0).position;

}//end initial touch

if(Input.GetTouch(0).phase == TouchPhase.Ended)

{

v2Current = Input.GetTouch(0).position;

Vector2 tempVector = v2Current - v2Previous;

fTouchDelta = tempVector.magnitude;

//Do the thing you wanted to do with the information

}//end finish touch

}//end touchCount == 1

}//end Update()

*Give an example of how to capture a two-touch input (pseudo-code or actual code):*

public class ZoomGesture : MonoBehaviour {

public GameObject gameObjectToRotate;

private Vector2 v2Current;

private Vector2 v2Previous;

private float fTouchDelta;

public float fComfortZone;

public Camera myCamera;

public float zoom = 5f;

public float minZoom = 10f;

public float maxZoom = 40f;

private float angle;

// Update is called once per frame

void Update () {

if(Input.touchCount == 2 &&

Input.GetTouch(0).phase == TouchPhase.Moved &&

Input.GetTouch(1).phase == TouchPhase.Moved)

{

v2Current = Input.GetTouch(0).position - Input.GetTouch(1).position;

v2Previous =

(Input.GetTouch(0).position - Input.GetTouch(0).deltaPosition) -

(Input.GetTouch(1).position - Input.GetTouch(1).deltaPosition);

fTouchDelta = v2Current.magnitude - v2Previous.magnitude;

angle = Vector2.Angle(v2Previous, v2Current);

//Do the thing you want to do with the information you have.

}

}

}

*Give an example of how to capture a simple gesture, like a pinch (pseudo-code or actual code):*

public class ZoomGesture : MonoBehaviour {

public GameObject gameObjectToRotate;

private Vector2 v2Current;

private Vector2 v2Previous;

private float fTouchDelta;

public float fComfortZone;

public Camera myCamera;

public float zoom = 5f;

public float minZoom = 10f;

public float maxZoom = 40f;

private float angle;

// Update is called once per frame

void Update () {

if(Input.touchCount == 2 &&

Input.GetTouch(0).phase == TouchPhase.Moved &&

Input.GetTouch(1).phase == TouchPhase.Moved)

{

v2Current = Input.GetTouch(0).position - Input.GetTouch(1).position;

v2Previous =

(Input.GetTouch(0).position - Input.GetTouch(0).deltaPosition) -

(Input.GetTouch(1).position - Input.GetTouch(1).deltaPosition);

fTouchDelta = v2Current.magnitude - v2Previous.magnitude;

angle = Vector2.Angle(v2Previous, v2Current);

if(angle > 0.1)

{

//Debug.Log("Rotation.");

//Debug.Log(Vector3.Cross(v2Current, v2Previous));

if(Vector3.Cross(v2Current, v2Previous).z < 0)

{

//Debug.Log("Counter Clockwise.");

gameObjectToRotate.transform.Rotate(Vector3.up, angle \* -1);

}

else

{

//Debug.Log("Clockwise.");

gameObjectToRotate.transform.Rotate(Vector3.up, angle);

}

}

if(Mathf.Abs(fTouchDelta) > fComfortZone)

{

//Debug.Log("zoom detected");

if(fTouchDelta > 0)//Zoom In

{

Debug.Log("Zoom In.");

myCamera.fieldOfView =

Mathf.Clamp(

Mathf.Lerp(myCamera.fieldOfView, myCamera.fieldOfView - Mathf.Abs(fTouchDelta)\*zoom, Time.deltaTime\*zoom),

minZoom, maxZoom);

}

else //Zoom Out

{

Debug.Log("Zoom Out.");

myCamera.fieldOfView =

Mathf.Clamp(

Mathf.Lerp(myCamera.fieldOfView, myCamera.fieldOfView + Mathf.Abs(fTouchDelta) \* zoom, Time.deltaTime \* zoom),

minZoom, maxZoom);

}

}

}

}

}