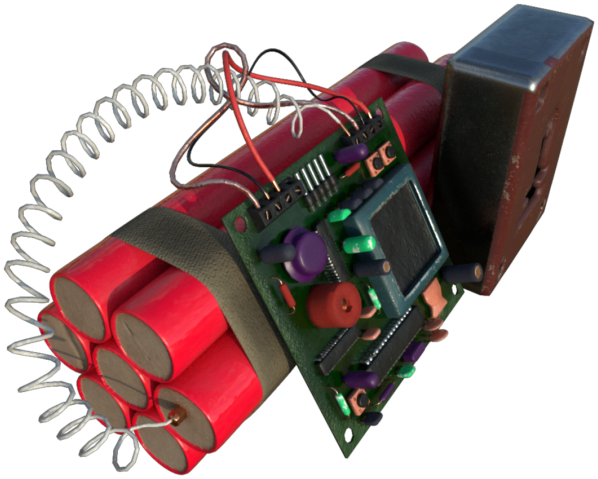
RTGD Workshop 1: BombSquadVR

Tom Corbett <tcorbett@andrew.cmu.edu>
Ben Scott <bescott@andrew.cmu.edu>



September 13, 2017

Abstract

Today's class is going to be very different. You're going to have to work with your new teammates to defuse a bomb. If you run out of time or if your bomb goes off, you fail the class. [1] [2]

[1] it's like those escape games but the prize is your academic career
[2] also just kidding you aren't going to fail the class

Introduction

Being able to work together effectively towards design goals is crucial. This workshop is intended to be a short tour of this brand of teamwork: You'll have to form consensus about shared goals and integrate your work. While some tasks will be technical, it's mostly about working together, so don't worry about "letting the team down" if you don't have hard skills. We have completed work for if the tasks are too technical for some people. [3]

Materials

You'll need some files and software tools to disarm this virtual bomb. If you came unprepared you can use the class laptops and take turns.

- Workshop Files: everyone should have downloaded the project
- Unity 2017.1f1: everyone should have this installed already
- Git 2.7.0: everyone will see why they should use git in the next hour
- Sublime Text 3: everyone should be able to edit simple text files [4]
- 3DS Max 2018: artists should have this, Maya, Rhino just this once [5]
- Audacity 2.1.3: sound designers probably already have this

[3] if you fall, I will catch you, I'll be waiting, Time after `fixedDeltaTime`

[4] real programmers are to use vim and must uninstall Visual Studio

[5] programs like AutoCAD and Rhino won't be suitable in the future

Procedure

This will be fast and difficult, and you're going to have fun (or else!) Everyone will be busy with their own work, and the time limit is grisly. Lots of things can cause a bomb to go off, so here are some ground rules:

- the bomb will detonate at 11:30 sharp, be ready well in advance
- the producer has to estimate when the team will be done with tasks
- the best time wins unless the team goes longer than their estimate
- one of you must manually insert the key to defuse the bomb, or else
- you can't just make an animation that inserts the key into the bomb
- you are not allowed to easily and flawlessly defuse bombs like that
- the bomb will detonate if you touch it at all before it's armed
- the bomb will detonate if the key touches the edges of the lock
- the bomb will detonate if the key is used when the display isn't green
- the bomb will detonate if the project files aren't well-organized
- the bomb will detonate if you even think about the Unity Asset Store
- the bomb will detonate if the main scene isn't hue shifted right
- the bomb will detonate if there's no tacky 80s rock ballad playing
- the bomb will detonate if the controller isn't implemented nicely
- the bomb will detonate if the artist's key isn't the right shape

If you get stuck on something you might need to fend for yourself. It's your responsibility to tell your producer if you can't do something, and then it's the producer's job to get the finished asset from the TA.

You don't have to use `git`, but it might keep you from blowing up. Described below are two approaches to integrating your work in Unity. You can use Unity Collab, but you'll be on your own and you might cry. It's like an informally specified and failure prone copy of `git`. You could also use the GitHub desktop tool if the command line is too hard.

Using `git` for Unity Development

- navigate to the directory where you want to put your files via `cd`
- find the link on github (green button) and clone files via `clone`
- do some work and check your changes via `status`
- add all the things you've changed via `add .`
- commit to your local branchwith a message via `commit -m`
- send your changes back to the master branch via `push`

```
~ $ git clone https://github.com/evan-erdos/bomb-squad-workshop.git
~ $ cd rtgd-bombsquad-workshop/
~ $ git status
~ $ git add .
~ $ git commit -m "I made all the changes"
~ $ git push
```

Using `*.unitypackages` Instead

- open up Unity and find the Project tab, select one or many files
- put those files into a new folder, and don't leave it in `Assets`
- when a package is imported it will appear wherever you left it
- right click and select "Export as Package", and see what's included
- anything that contains a script might try to export all scripts
- go ahead and do not export every single script in the entire project
- save the file, and send it to whomever you think needs to have it

Producers

The producer will delegate tasks and facilitate communication.

1. read the assignment and create a plan of action and a schedule
2. explain everyone's roles and delegate tasks to the team clearly
3. facilitate communication and transfer files between teammates
4. schedule usage of the resources and identify emergent problems
5. gage progress towards the goal and submit deliverables in time
6. Bonus: manage a trello board / github project board while working

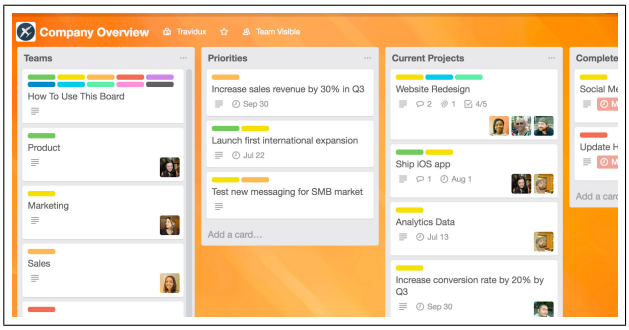
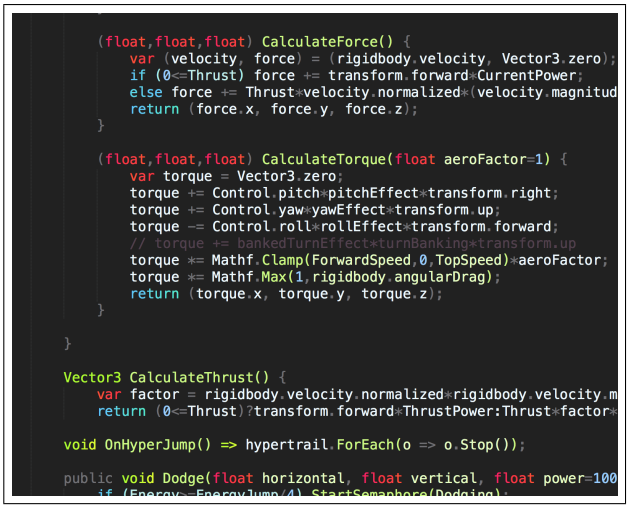


Figure 1: a Trello Board

Programmers

The programmer will implement the controller in an ergonomic way.

1. find `IController.cs` and `Controller.cs`
2. scour `Bomb.cs` and `IBomb.cs` for clues
3. implement `Controller` to satisfy the interface
4. think about ways to stabilize or otherwise aid the user
5. commit work via `git` or make a `*.unitypackage`
6. tell your producer where to find your work once you're done
7. Bonus: use a rendertexture to give the user a better view



```
        (float,float,float) CalculateForce() {
            var (velocity, force) = (rigidbody.velocity, Vector3.zero);
            if (0<=Thrust) force += transform.forward*CurrentPower;
            else force += Thrust*velocity.normalized*(velocity.magnitud
            return (force.x, force.y, force.z);
        }

        (float,float,float) CalculateTorque(float aeroFactor=1) {
            var torque = Vector3.zero;
            torque += Control.pitch*pitchEffect*transform.right;
            torque += Control.yaw*yawEffect*transform.up;
            torque -= Control.roll*rollEffect*transform.forward;
            // torque += bankedTurnEffect*turnBanking*transform.up
            torque *= Mathf.Clamp(ForwardSpeed,0,TopSpeed)*aeroFactor;
            torque *= Mathf.Max(1,rigidbody.angularDrag);
            return (torque.x, torque.y, torque.z);
        }

    }

    Vector3 CalculateThrust() {
        var factor = rigidbody.velocity.normalized*rigidbody.velocity.m
        return (0<=Thrust)?transform.forward*ThrustPower:Thrust*factor*

    void OnHyperJump() => hypertrail.ForEach(o => o.Stop());

    public void Dodge(float horizontal, float vertical, float power=100
        if (Energy>=EnergyJump/4) StartCoroutine(Dodging());
```

Figure 2: some important code

3D Modelers

The modeler will create a key which fits the lock on the bomb.

1. model a key which fits the size and shape of the bomb's keyhole
2. export as `*.fbx` and import the model into Unity
3. create a material, make it lime green, and apply it to the key
4. commit work via `git` or make a `*.unitypackage`
5. tell your producer where to find your work once you're done
6. Bonus: make some scenery props for decor, try UV mapping

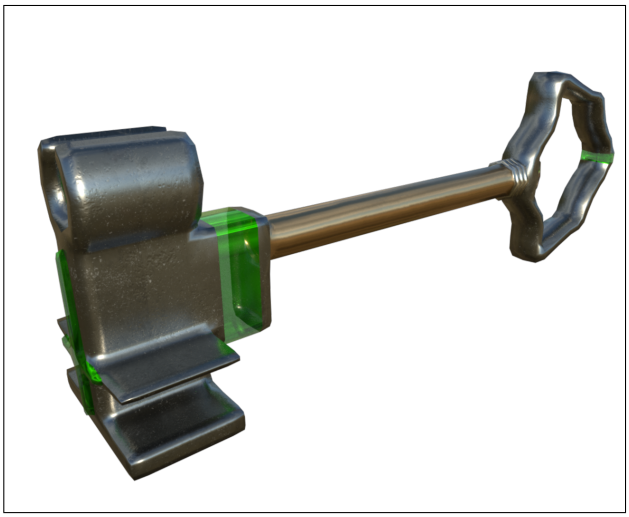


Figure 3: The Bomb's default Key

2D Artists

The artist will create a display material for the bomb.

1. make some new files in Photoshop or whatever you have
2. make files for albedo, metallic, and emissive maps
3. play with painting different colors into different maps
4. make a bright, readable background material for the bomb display
5. put some kind of icon or text to let the defuser know when to try
6. commit work via `git` or make a `*.unitypackage`
7. tell your producer where to find your work once you're done
8. Bonus: Get Substance Painter and repaint the controllers

Figure 4: fun with Substance Painter

Game Designers

The designer will assemble the scene with the Player and the bomb.

1. open Unity and find the bomb amidst the empty scenes
2. fun fact, you can drag all the scenes in at once
3. drag the bomb into the Project tab to create the prefab
4. create a new scene and save it somewhere sensible
5. the bomb will detonate unless you hue shift the scene red
6. find a way in the postprocessing effects to hue shift
7. commit work via `git` or make a `*.unitypackage`
8. tell your producer where to find your work once you're done
9. Bonus: play around with the skybox and bake lighting

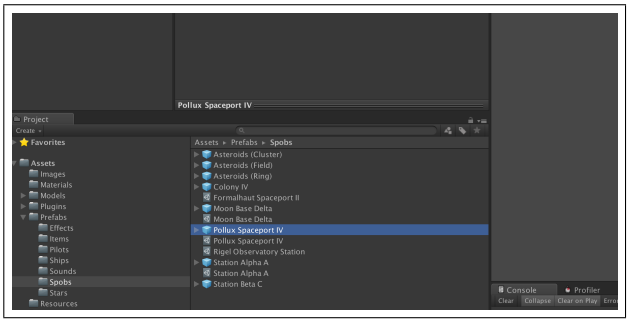


Figure 5: making a prefab in Unity

Sound Designers

The sound designer will find or record a background song for the bomb.

1. open Unity and find the `Sounds` directory
2. find or make a warning sound for when the bomb is triggered
3. drag it into `Sounds`, check your import settings
4. drag the sound asset into the scene, and make sure it's 3D
5. add an `OculusSpatializer` component to the sound
6. drag the sound object back into the editor to make a prefab
7. the bomb will detonate unless tacky 80s music is played at it
8. find tacky 80s music on the internet, and import it into Unity
9. make a new prefab as before with this sound, and set it to loop
10. commit work via `git` or make a `*.unitypackage`
11. tell your producer where to find your work once you're done
12. Bonus: Make a Mixer so you don't have to listen to awful music

Figure 6: tacky 80s music

Discussion

What fun we've had today, doing all what's described above!

1. What can we glean from this about designing VR interactions?
2. What were some of the adverse effects of VR? (Figure 7)
3. How did the teams who used `git` fare? The ones that didn't?
4. What worked? What didn't? Why? What if the bombs had been real?
5. What unexpected consequences came out of the iteration process?

Figure 7: adverse effects of virtual reality