FINAL REPORT
DADIU: Danske Akademi For Digital, Interaktiv Underholdning

Ruben Kolodny Lopez
University of Copenhagen, Department of Computer Science
zmv615@alumni.ku.dk
+45 50 25 51 90

1. Evaluations and Technical Documents

Evaluations at DADIU were made at the end of our semester (when we finished our month-long production of our "graduation game"), and was mainly focused on this final, month-long period in which we worked as a team (made-up of managers, directors, programmers, artists, etc. -- much like in the game-industry). However, for the evaluations, we were not evaluated as a whole team. Rather, we were divided by our competences, meaning: the programmers got split into one group, the artists in another group, the managers in another, and so forth.

For the programmers's evaluation we had a Software Engineer come in and talk to us. He not only asked us questions but also watched a short presentation of each team's game, in which we discussed some of the programming tasks that were required for our game. We discussed our approach to these tasks, some of the challenges we faced, and took questions from other programers (from other groups) to generate discussion and spark questions for our evaluator, the Software Engineer.

Technical Documents (risks, pitch, game mechanics, etc):

RISK ANALYSIS

ID	PROBABILITY	IMPACT	OWNER	STATUS	LAST REVISED	NAME	DESCRIPTION	MITIGATION STRATEGY
1	Medium	Very high	Code	Pending	4/12/2013	File Size	Get the game under the 50mb limit without severly damageing the quality	Keep a constant track of the game's size. Explore and test compression options for graphics, sound, etc.
2	Medium	Medium	Code	Active	2/12/2013	Merge	Overwrites happening when the programmers work on the same scripts unknowingly	Lead programmer checks all commits before they are put on the master branch in GIT.
3	High	Medium	Production	Active	5/12/2013	Exhaustion	Some team members have been working very late and are tired and might make errors because of it	make sure people take breaks aswell as asking them to take naps when necessary
4	Low	Very high	Production	Active	3/12/2013	Technical Failure	Computer Errors,No internet,No electricity and the like	All files are kept on external servers minimizing the loss caused by technical failure
5	Very high	Medium	Code	Active	4/12/2013	Performance	If the game becomes to heavy and slows FPS too much it will influence gameplay negatively	Monitor the FPS everytime a new build is made and optimize if it slows too much
6	High	High	Code	Active	8/11/2013	GitHub	The external GitHub server and cause loss of data	Every computer has a copy of the game stored locally
7	Medium	Very high	Design	Retired	20/11/2013	Lift	making sure that the lift does not conflict with other mechanics	Made a new design that excludes lift as a mechanic
8	Low	High	Sound	Active	5/11/2013	Sound Storage	The sound files are to big to run through git meaning if the computer fails all data is lost	All the sounds are constantly updated and kept on two computers. A stationary and a laptop
9	Very high	Medium	Code	Retired	3/11/2013	Mechanim	Choosing humanoid animation type resulted in faulty animation	Changed the animation type to generic

GAME AND STORY SYNOPSIS

Punish Panda is a game with lots of bloody violence. Kill kill kill!! Enjoy the guilty delicious pleasure of slapping a panda so hard the blood spats. Punish them, because you can – and because it feels fucking awesome!!

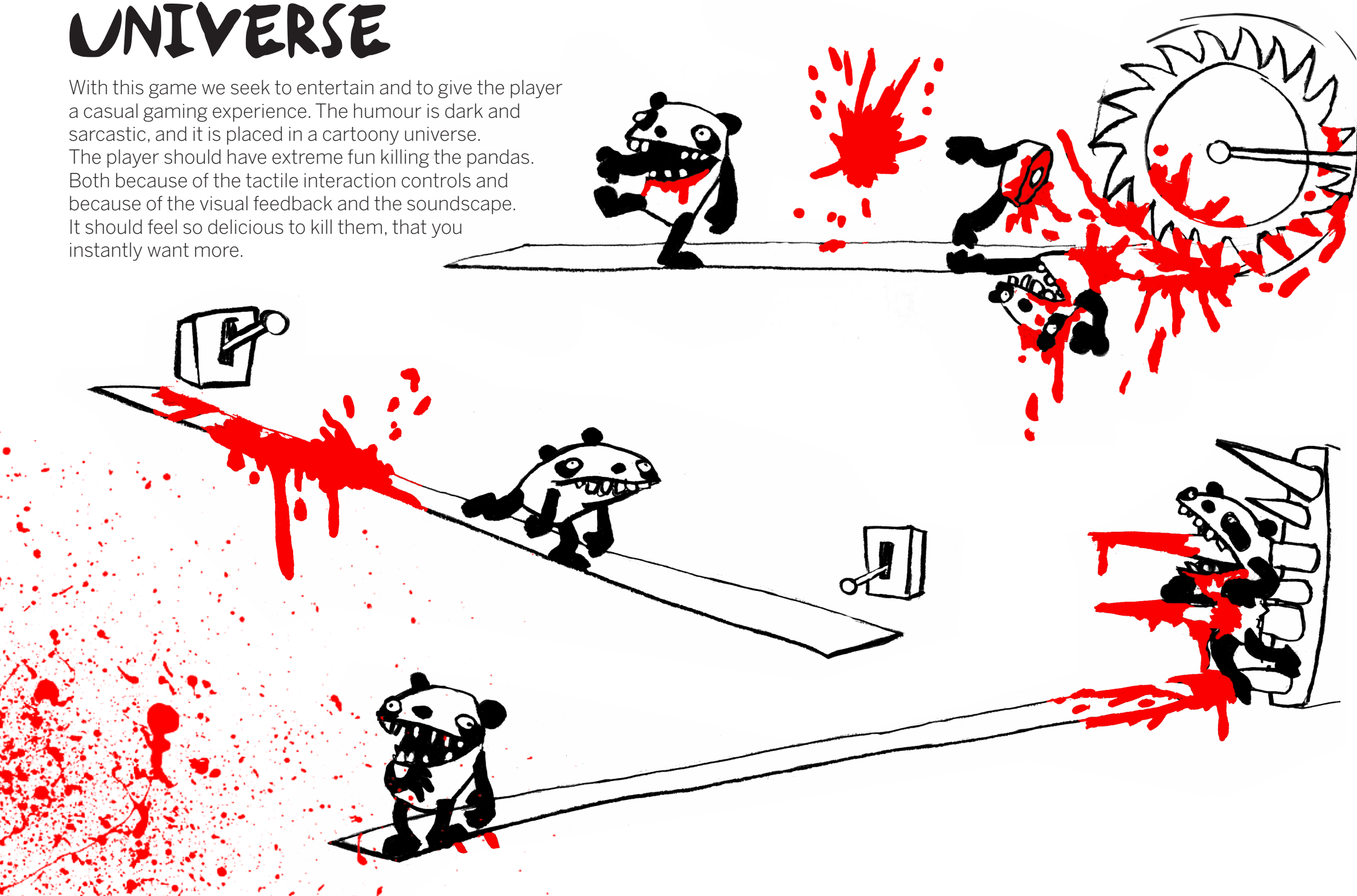
It is time to reveal the ugly truth! Kill the pandas, before they take over the world.

Pandas are not cute, fluffy or endangered. It is a scam!

Pandas are vulgar, selfish and evil creatures craving world domination

UNIVERSE

With this game we seek to entertain and to give the player a casual gaming experience. The humour is dark and sarcastic, and it is placed in a cartoony universe.
The player should have extreme fun killing the pandas. Both because of the tactile interaction controls and because of the visual feedback and the soundscape. It should feel so delicious to kill them, that you instantly want more.



GAMEPLAY

DESCRIPTION

Punish Panda is a side-viewed puzzle platformer, where you have to kill pandas. There can be between 1-4 pandas in a chamber. The player has to guide the pandas through mazes of deathtraps to commit perfect panda kills before they escape. When all pandas in a level are killed, the chamber is cleared. Each level has one or more 'perfect' kill traps that will help the player achieve a higher score. The perfect kill traps are prompted in the beginning of each level, so the player gets a clue of how to complete the level to perfection.
The player can also improve his score with combo kills, where 2 or more pandas are killed at the same time.

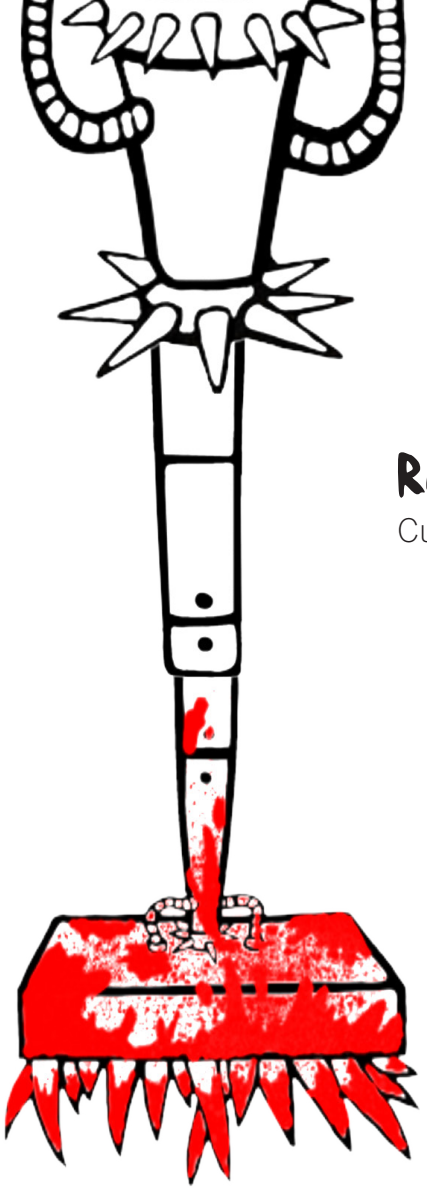
PROGRESSION

The quicker the player completes the level, with as many 'perfect' kills and combo kills as possible, determines the level score. This score will result in a grading of 1-3 bloodsplats. The player can replay all levels to improve his score and/or get 3 bloodsplats in all levels. The player unlocks next level by completing a level with a score of 1-3 bloodsplats. Time as a variable will ensure that score are diverse, and the player will feel that he can always improve himself.

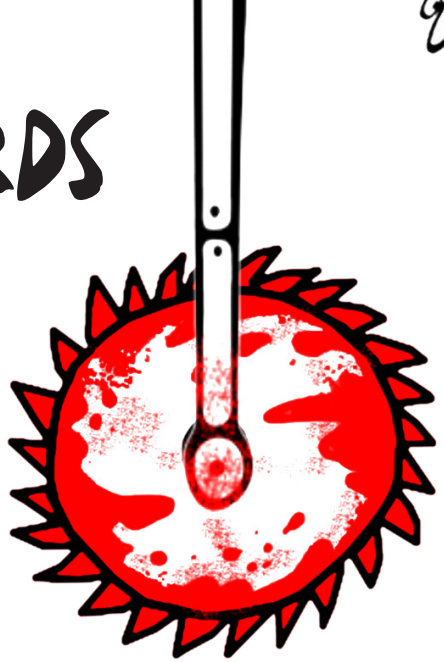
GAME MECHANICS

Players can affect the movements of the pandas with on of three input controls.
Tap and hold in front of a panda – holds one panda. When released, the panda will continue in the same path as before.
Swipe to slap - Slaps the panda making it go in the opposite direction. When a panda gets slapped it also spews blood.
Tap and hold on designated hotspot when Pandas walk over the player's finger they will bounce off it. As a design principle you can only use two fingers at the same time. This is because the game should be playable while holding the tablet (fx. while playing on the bus). There are two main gameplay mechanic groups; Mechanics and Hazards.

HAZARDS



POUNDERS
Crushes pandas

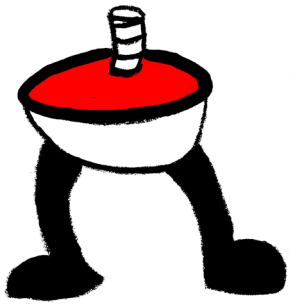


ROUNDSAW
Cuts pandas in half

ELECTRICITY
Shocks pandas

SPIKES
Spears pandas. Can be placed on side of the screen and on floors.

THROWING STARS
Kills the pandas. Can kill multiple pandas in a row. Can be blocked by the finger.



GAMEPLAY MECHANICS

Side wall Buttons – Pandas walk into a wall and turn on/off button as they change walking direction – these buttons are timed, so it changes its state back after a certain time.
Levers – allows pandas to turn hazards on/of when they walk by them.
Pressure pads - allows pandas to turn hazards on when they stand on the pads. They turn off when the pandas are not standing on them.
Trapdoors - opens up a hole in the floor when activated

TARGET GROUP

Punish Panda is for those who love sarcastic humour, and who can have an ironic approach to serious subjects. What is provocative to many is sweet music in this person's ears. The type who watches family Guy and The Simpsons, one who quotes Southpark. One who checks news online, who reads Politikken.dk, mostly just the headliners – and always the comic strips. This target group does not come from a specific social class.

COMPETITOR ANALYSIS

Below is a sample of the key competitors. What is the deciding factor in relating these games to Punish Panda is that they are all side-view puzzle games which utilize humor.

GAME NAME	 PUNISH PANDA	 CUT THE ROPE	 ABOUT LOVE, HATE	 BAND TOGETHER	 RAGDOLL BLASTER 3
Developer	Burnt Flamingo	ZeptoLab	Tobias Bilgeri	Backflips Studios	Backflips Studios
Platform	Android Tablet	Android/iOS Tablets	iPad	iPad	iPadEstimated
Downloads	0	300 mio.	Unknown	Unknown (300 mio. all games)	Unknown (300 mio. all games)
Genre	Puzzle Platformer	Physics Puzzle	Puzzle Platformer	Puzzle Platformer	Physics Puzzle
Universe	Black/White Cartoony	Cartoony, abstract, historical	Hand drawn Caves	Semi-Realistic Cardboard	Cartoony enviorment settings
Humor Presence	High	High	Medium	Low	High
Cost	Free	2.99$	3.99$	0.99$	Free
Metascore	None	93	79	79	80
Similarities with PP		Replayability	Puzzle mindset	Deathtraps	Character torture
		Side-view 2D levels	Side-view 2D levels	Side-view 3D levels	Side-view 3D levels
PP different by		Having morbid humor	Being timebased	Control scheme	Having limited characters
				Having killing as a goal	

2. Personal Contributions and Source Code

Before I delve into the source code that I focused on most, its worth mentioning that all the programmers on my team (including myself) worked on each others's scripts -- even if our role was only to work on 1 specific task (or script) at a time. This is because edits, debugging, and scalability (in other scripts) was constantly needed in order for all the scripts to cooperate as a whole, in the game. That being said, my personal contributions were spread throughout the project. However, the scripts that were mainly made by me include:

BloodOnSlap.cs -- a script that projects blood particles in the same direction of a swipe-event (i.e. when the user swipes (or slaps) a Panda). Particle emission is based on the Panda's position, and its vector-direction corresponds with the blood splat to the wall.

PauseMenuManager.cs -- a manager for the pause menu, which includes handling of hint/tutorial screens, retrieval of level data, button event handling, pausing events, etc.

PauseGame.cs -- handles the pausing and resuming of the game by changing the game's timescale (to zero). It also calls our input-handler and tells it to ignore in-game input events (when paused).

LevelIconManager.cs -- a manager for the level icon's in the "levels screen" which checks if a level is locked/unlocked, has earned 1,2 or 3 stars, and will change its appearance/functionality based on these statistic checks.

MainMenuSaws.cs -- a script for making the main menu buttons (saws) highly interactive: for example, making the object (i.e. button) spin, and emit dismembered pandas when we press and hold the button.

RemoveAfterTime.cs -- simple script for removing (deleting) certain objects after they have been instantiated (i.e. cloned) so that we can save resources and improve performance. It is used for objects such as blood particles and dismembered panda.

GuiButtonAlternator.cs -- another simple script that alternates (i.e. toggles) the texture of a button when it is switched on or off.

SOURCE CODE:

BloodOnSlap.cs:

```
using UnityEngine;
using System.Collections.Generic;

public enum Version {v1, v2}
public class BloodOnSlap : MonoBehaviour {

    /*******************************************************
     *   This takes the BloodSplat(particle) object
     *   and instantiates it based on Panda's position.
     *   Function is called on a Slap (in PandaAI).
     *******************************************************/

    public GameObject PandaMouthFront; //use for start position of emmision
    public GameObject PandaMouthBack; //use for turn-slap
    public Version particleVersion;
    public GameObject particleV2;

    private string objectName;
    private float timeStamp1;
    private float timeStamp2;
```

```
// 3D vector controlling the direction of the blood particle object
private Vector3 projectionDirection = Vector3.right;

[EventHookAttribute("Slap")]
[SerializeField] List<AudioEvent> slapAudioEvents = new List<AudioEvent>();

void Start ()
{
    if(particleVersion == Version.v1)
    {
        objectName = "particle_slap_v1";
    }
    else
    {
        objectName = "particle_slap_v2";
    }

}

//Method for emmiting blood in the same direction(and angle) as slap.
//NOTE: we use this when panda is slapped in the BACK.
public void EmmitSlapBlood(Vector2 slapDirection)
{
    float diff;

    for(int i = 0; i < slapAudioEvents.Count; ++i)
    {

        HDRSystem.PostEvent(gameObject, slapAudioEvents[i]);
    }

    // set the angle of the splat to be the same in both XY and XZ planes
    projectionDirection.x = slapDirection.x;
    projectionDirection.y = slapDirection.y;
    projectionDirection.z = Mathf.Abs(slapDirection.x);


    timeStamp1 = Time.realtimeSinceStartup;

    diff = timeStamp1 - timeStamp2;
    diff = Mathf.Abs (diff);

    if(diff > 0.1)
    {
        //Object instantiates facing the Z-axis direction
        Instantiate(particleV2 ,PandaMouthFront.transform.position, Quaternion.LookRotation(projectionDirection));
    }
}

//NOTE: use this when panda is slapped from its FRONT (and does a turn)
public void EmmitSlapBloodOnTurn(Vector2 slapDirection)
{
    float diff;

    for(int i = 0; i < slapAudioEvents.Count; ++i)
    {

        HDRSystem.PostEvent(gameObject, slapAudioEvents[i]);
    }

    // set the angle of the splat to be the same in both XY and XZ planes
    projectionDirection.x = slapDirection.x;
    projectionDirection.y = slapDirection.y;
    projectionDirection.z = Mathf.Abs(slapDirection.x);

    timeStamp2 = Time.realtimeSinceStartup;

    diff = timeStamp1 - timeStamp2;
    diff = Mathf.Abs (diff);
```

```
        if(diff > 0.1)
        {
            //Object instantiates facing the Z-axis direction
            Instantiate(particleV2,PandaMouthBack.transform.position, Quaternion.LookRotation(projectionDirection));
        }


    }
}
```

PauseMenuManager.cs:

```
using UnityEngine;
using System.Collections;

public class PauseMenuManager : MonoBehaviour {

    PauseGame pausegame;

    public GameObject PauseMenu;
    public GameObject PauseTint;
    public GameObject HintScreen;
    public GameObject HintObj;
    public GameObject PauseAndReset;
    public GameObject WhiteTint;
    public GameObject LevelNumberLabel;
    private UILabel levelNumberComponent;

    private UITexture textureComponent;
    private TweenAlpha hintAlphaComponent;
    private TweenAlpha tintAlphaComponent;
    private TweenAlpha whiteTintAlphaComponent;

    private Texture2D HintTexture;
    private Texture2D TutorialTexture;

    private bool MenuIsActive;
    private TweenPosition MenuPosition;

    void Start()
    {
        MenuPosition = PauseMenu.GetComponent<TweenPosition>();
        pausegame = GetComponent<PauseGame>();

        //get Level # for label
        levelNumberComponent = LevelNumberLabel.GetComponent<UILabel>();
        int lvlNumInt = InstanceFinder.LevelManager.CurrentLevelIndex;
        lvlNumInt ++;
        string levelNumString = lvlNumInt.ToString();
        levelNumberComponent.text = levelNumString;

        //get components
        textureComponent = HintObj.GetComponent<UITexture>();
        hintAlphaComponent = HintObj.GetComponent<TweenAlpha>();
        tintAlphaComponent = PauseTint.GetComponent<TweenAlpha>();
        whiteTintAlphaComponent = WhiteTint.GetComponent<TweenAlpha>();

        if(!InstanceFinder.GameManager.debugMode)
        {
            //get start-tutorial texture:
            if (Localization.instance.currentLanguage == "English")
            {
                HintTexture = InstanceFinder.LevelManager.CurrentLevel.HintscreenTexture;
                TutorialTexture = InstanceFinder.LevelManager.CurrentLevel.TutorialTexture;
            }
            else
```

```
            {
                HintTexture = InstanceFinder.LevelManager.CurrentLevel.DanishHintscreenTexture;
                TutorialTexture = InstanceFinder.LevelManager.CurrentLevel.DanishTutorialTexture;
            }

            //set start-tutorial texture to component
            textureComponent.mainTexture = TutorialTexture;
            //set it to its native-dimensions
            if(textureComponent.mainTexture != null)
            {
                textureComponent.width = TutorialTexture.width;
                textureComponent.height = TutorialTexture.height;

                StartCoroutine(startTutorial ());
            }
        }
    }

    public void OnPauseClick()
    {
        //Enable Screen tint
        PauseTint.SetActive(true);
        tintAlphaComponent.PlayForward();
        //Enable PauseMENU
        PauseMenu.SetActive (true);
        MenuIsActive = true;

    }

    public void OnResumeClick()
    {
        tintAlphaComponent.Reset();
        PauseTint.SetActive(false);
        MenuIsActive = false;
    }

    public void OnHintClick()
    {
        //reset PauseMenu Position
        MenuPosition.Reset();

        //set menu-hint texture
        textureComponent.mainTexture = HintTexture;
        if(textureComponent.mainTexture != null)
        {
            textureComponent.width = HintTexture.width;
            textureComponent.height = HintTexture.height;

            HintScreen.SetActive(true);
            PauseMenu.SetActive(false);
            tintAlphaComponent.Reset();
            PauseTint.SetActive(false);
            WhiteTint.SetActive(true);

            hintAlphaComponent.PlayForward();
            whiteTintAlphaComponent.PlayForward();
        }
        else
        {
            Debug.Log ("A Hint picture is NOT setup for this level!");
        }

    }

    //called "onPress"
    public void OnHintReturnClick()
    {
        hintAlphaComponent.Reset();
        whiteTintAlphaComponent.Reset ();
        WhiteTint.SetActive(false);
```

```
        HintScreen.SetActive(false);

        PauseAndReset.SetActive (true);

    }

    public void OnLevelsClick()
    {
        //Unpause game to get the normal TimeScale back
        pausegame.ResumeGame();
        InstanceFinder.LevelManager.LoadLevelsMenu();
    }

    public void OnMainMenuClick()
    {
        //Unpause game to get the normal TimeScale back
        pausegame.ResumeGame();
        InstanceFinder.LevelManager.LoadMainMenu();
    }

    public void DisablePauseMenu()
    {
        if(MenuIsActive == false)
            PauseMenu.SetActive (false);
    }

    //for showhing tutorial screen/animation at levelstart
    IEnumerator startTutorial()
    {
        // wait one update for data initialization
        yield return null;

        pausegame.TutorialPause();
        PauseAndReset.SetActive (false);


        WhiteTint.SetActive(true);
        HintScreen.SetActive(true);

        hintAlphaComponent.Play();
        whiteTintAlphaComponent.Play();
    }

    IEnumerator ExitTutorial()
    {
        hintAlphaComponent.PlayReverse();
        whiteTintAlphaComponent.PlayReverse ();

        float fadeOutTime = Time.realtimeSinceStartup + 1;
        while (Time.realtimeSinceStartup < fadeOutTime)
        {
            yield return 0;
        }
        hintAlphaComponent.Reset();
        whiteTintAlphaComponent.Reset ();
        WhiteTint.SetActive(false);
        HintScreen.SetActive(false);
    }
}
```

PauseGame.cs:

```
using UnityEngine;
using System.Collections.Generic;

public class PauseGame : MonoBehaviour {

    private float savedTimeScale;
```

```
    private GameObject inputHandler;
    private InputHandler inputScript;

    [SerializeField] [EventHookAttribute("PauseGame")]
    List<AudioEvent> pauseGameEvent;

    [SerializeField] [EventHookAttribute("ResumeGame")]
    List<AudioEvent> resumeGameEvent;

    void Start ()
    {
        savedTimeScale = Time.timeScale;
        inputScript = InputHandler.instance;
    }

    public void StopTime()
    {
        Time.timeScale = 0;
        InputHandler.instance.PausedGame();

        //PAUSE AUDIO ALSO??
        for(int i = 0; i < pauseGameEvent.Count; ++i)
        {
            HDRSystem.PostEvent(gameObject, pauseGameEvent[i]);
        }
    }

    public void TutorialPause()
    {
        Time.timeScale = 0;


        //PAUSE AUDIO ALSO??
        for(int i = 0; i < pauseGameEvent.Count; ++i)
        {
            HDRSystem.PostEvent(gameObject, pauseGameEvent[i]);
        }
    }

    public void ResumeGame()
    {
        Time.timeScale = savedTimeScale;
        InputHandler.instance.UnpausedGame();

        for(int i = 0; i < resumeGameEvent.Count; ++i)
        {
            HDRSystem.PostEvent(gameObject, resumeGameEvent[i]);
        }
    }

    public void RestartLevel()
    {
        InstanceFinder.LevelManager.Reload();
    }
}
```

LevelIconManager.cs:

```
using UnityEngine;
using System.Collections;

public class LevelIconManager : MonoBehaviour {

    //Handler for the LEVELS SCREEN that checks how many
    //stars should be shown on a Level Icon.

    public int LevelNumber;
```

```
public bool isUnlocked;

public GameObject first;
public GameObject second;
public GameObject third;
private UISprite firstSprite;
private UISprite secondSprite;
private UISprite thirdSprite;

//More functionality:
public GameObject unlockedLabel;
public GameObject lockedLabel;
private UILabel Number;

void Start () {
	firstSprite = first.GetComponent<UISprite>();
	secondSprite = second.GetComponent<UISprite>();
	thirdSprite = third.GetComponent<UISprite>();

	string levelNumberString = LevelNumber.ToString();
	Number = unlockedLabel.GetComponent<UILabel>();
	Number.text = levelNumberString;
	Number = lockedLabel.GetComponent<UILabel>();
	Number.text = levelNumberString;


	var levels = InstanceFinder.LevelManager.CurrentWorld.Levels;
	int stars;

	if(LevelNumber <= levels.Count )
	{
		isUnlocked = levels[LevelNumber-1].UnlockedLevel;
		//STAR CALCULATION:
		stars = PunishPanda.Game.ScoreCalculator.Stars(levels[LevelNumber - 1], levels[LevelNumber-1].HighScore);
		//Debug.Log ("Level:"+LevelNumber+" has "+stars+" stars");

		if(isUnlocked == true)
		{
			if(stars == 1)
			{
				show1star();
			}
			else if(stars == 2)
			{
				show2stars();
			}
			else if(stars == 3)
			{
				show3stars ();
			}
		}
	}
	else {
		//level doesnt exist in build so just lock it
		LockLevel();
	}

	if(isUnlocked == false)
		LockLevel ();
}

public void show1star()
{
	firstSprite.spriteName = "bloodStarSplat01";

}

public void show2stars()
{
	firstSprite.spriteName = "bloodStarSplat01";
```

```
        secondSprite.spriteName = "bloodStarSplat02";
    }

    public void show3stars()
    {
        firstSprite.spriteName = "bloodStarSplat01";
        secondSprite.spriteName = "bloodStarSplat02";
        thirdSprite.spriteName = "bloodStarSplat03";
    }

    public void LockLevel()
    {
        //change appearence
        unlockedLabel.SetActive(false);
        lockedLabel.SetActive (true);

        //disable collider (so we cant click button)
        (gameObject.GetComponent(typeof(Collider)) as Collider).enabled = false;

    }

    public void UnlockLevel()
    {
        //change appearence
        lockedLabel.SetActive (false);
        unlockedLabel.SetActive (true);

        //enable collider
        (gameObject.GetComponent(typeof(Collider)) as Collider).enabled = true;
    }
}
```

MainMenuSaws.cs:

```
using System.Collections.Generic;
using UnityEngine;
using System.Collections;

public class MainMenuSaws : MonoBehaviour {

    public GameObject sawObject;
    public SawTrap sawtrap;
    private UITexture textureComponent;
    public Texture2D e_playImage;
    public Texture2D e_achieveImage;
    public Texture2D e_unlocksImage;
    public Texture2D d_playImage;
    public Texture2D d_achieveImage;
    public Texture2D d_unlocksImage;

    private Quaternion originalPosition;

    public GameObject projectionPoint;

    [SerializeField] protected GameObject dismemberedPanda;

    [SerializeField] [EventHookAttribute("On Start Spin")]
    private List<AudioEvent> startSpinEvents = new List<AudioEvent>(1);

    [SerializeField]
    [EventHookAttribute("On End Spin")]
    private List<AudioEvent> endSpinEvents = new List<AudioEvent>(1);

    void Start () {


        OnEnable ();
```

```
        //save orginial rotational-position
        originalPosition = sawObject.transform.rotation;
        //add deggress:
        originalPosition = Quaternion.Euler(0, 0, -45);

        //adjusting orginal position so it spins perfectly back in place
        //NOTE: this works with acceleration = 20
        if (Localization.instance.currentLanguage == "English")
        {
            if(this.gameObject.name == "Achievements Button")
            {
                //45 to 19
                originalPosition = Quaternion.Euler(0, 0, -26);
            }
        }
        else
        {
            if(this.gameObject.name == "Achievements Button")
            {
                //45 to 7.47
                originalPosition = Quaternion.Euler(0, 0, -37.53f);
            }
        }

    }

    public void OnPress(bool isDown)
    {
        if(isDown)
        {
            sawtrap.ActivateTrap();
            HDRSystem.PostEvents(gameObject, startSpinEvents);
            StartCoroutine("emmitDismembered");
        }

        if(!isDown)
        {

            //also do all this the object gets disabled
            StopCoroutine("emmitDismembered");
            HDRSystem.PostEvents(gameObject, endSpinEvents);
            sawtrap.DeactivateTrap();
            sawObject.transform.rotation = originalPosition;

        }
    }

    IEnumerator emmitDismembered()
    {
        //delay before Instantiate
        yield return new WaitForSeconds(0.7f);
        Instantiate(dismemberedPanda, projectionPoint.transform.position, projectionPoint.transform.rotation);
        yield return new WaitForSeconds(Random.Range(0.2F, 0.5F));
        Instantiate(dismemberedPanda, projectionPoint.transform.position, projectionPoint.transform.rotation);
        yield return new WaitForSeconds(Random.Range(0.3F, 0.7F));
        Instantiate(dismemberedPanda, projectionPoint.transform.position, projectionPoint.transform.rotation);
        StartCoroutine("emmitDismembered");
    }

    //fix for multitouching saws
    void OnDisable()
    {
        StopCoroutine("emmitDismembered");
        sawtrap.DeactivateTrap();
        sawObject.transform.rotation = originalPosition;
        sawtrap.Reset ();

    }
```

```
    void OnEnable()
    {

        sawtrap.Reset();

        //get textureComponent
        textureComponent = sawObject.GetComponent<UITexture>();

        if (Localization.instance.currentLanguage == "English")
        {
            if(this.gameObject.name == "Play Button")
            {
                textureComponent.mainTexture = e_playImage;
            }
            else if(this.gameObject.name == "Achievements Button")
            {
                textureComponent.mainTexture = e_achieveImage;
                sawObject.transform.rotation = Quaternion.Euler(0, 0, 19);
            }
            else if(this.gameObject.name == "Unlocks Button")
            {
                textureComponent.mainTexture = e_unlocksImage;
            }

        }
        else
        {
            //selected language is Danish..
            if(this.gameObject.name == "Play Button")
            {
                textureComponent.mainTexture = d_playImage;
            }
            else if(this.gameObject.name == "Achievements Button")
            {
                textureComponent.mainTexture = d_achieveImage;
                sawObject.transform.rotation = Quaternion.Euler(0, 0, 7.47f);
            }
            else if(this.gameObject.name == "Unlocks Button")
            {
                textureComponent.mainTexture = d_unlocksImage;
                sawObject.transform.rotation = Quaternion.Euler(0, 0, 357);
            }

        }

    }

}
```

RemoveAfterTime.cs:

```
using UnityEngine;
using System.Collections;

public class RemoveAfterTime : MonoBehaviour {

    public float EventLength = 2f;

    void Start ()
    {
        if(this.gameObject.name == "particle_slap_v1(Clone)"    ||
           this.gameObject.name == "particle_slap_v2(Clone)" ||
           this.gameObject.name == "Menu_DismemberedRigidBody(Clone)" ||
           this.gameObject.name == "Menu_Dismembered(Clone)" ||
           this.gameObject.name == "blood_spray" )
        {
            Destroy(gameObject, EventLength);
```

```
        }
    }

}
```

---

3. The Computer Science Problems We Faced During Development:

Merging Problems (with our Version Control System):

For me, the most agitating problem throughout the project was when we lost or overwrote other programmers's work due to our poor understanding of the version control system (i.e using GIT to merge, pull, and push our branches). Although most of us had experience with GIT, we had never used GIT with Unity -- our 3D Game Development Software -- and was therefore unaware of the problems we would encounter.

Unlike our scripts (which were easy to merge), the GameObjects (within Unity) weren't always merge-able. The GameObjects would sometimes give us merging-conflicts, and we couldn't manually resolve them because the source code for the GameObjects was unreadable (i.e. it was in binary or a native-Unity-language). So in order for our changes to take effect, we were often forced to overwrite other people's edits or tweaks to a GameObject.

Our solution to this problem was to establish a rule (among the programmers) to set a "lock" on a certain GameObject if someone decided that they were going to make changes to it. That way you (and only you) were allowed to make changes to it, and so that when we committed the change to the Master Branch we wouldn't have to worry about a merging-conflict.

CONCLUSION:
What I Have Learned From Working In That Environment

The semester-long DADIU program has been an enlivening experience that has taught me how people of different competences (managers, directors, artists, programers,etc) can work together to deliver an awesome product through teamwork and organization. Having the opportunity to work in an environment that closely simulated a production in industry -- with deadlines, team meetings, frequent communication, etc -- was to me, most beneficial to my education.

For example, during development, an idea or feature would get pipelined through various competences: say, from Game Designer -> to Artist -> to CG artist -> then to me (the programmer), and then have it be implemented in the game. This required a mutual understanding from everyone involved, and could only be accomplished through cooperation and clear communication. These are the social-skills that were exercised in this environment.

I will really value the experience at DADIU, and I know it will help me in my future career path (where communication among other competences will be necessary).

4. Game Presentation:

If for any reason a personal demonstration cannot be arranged (because I'm on exchange and will not be in Copenhagen anymore in January), you can check out this 8 minute play-through of our game!
https://www.dropbox.com/s/tw42mk9ef5dwtx4PunishPanda_ingame.mp4