
PRODUCTIONS

Letter Storm
Iteration 3

Brought to you by **GGProductions**

Nabil Lamriben, Paul Pollack, David Lustig, Jeannie Trinh, James Raygor

Project Goal

Our goal is to provide children with a digital learning experience that is truly fun, entertaining, and educational, without sacrificing one aspect for the others.

Requirements

Requirements

Overview

- User stories
- Functional requirements
- Nonfunctional requirements
- How we track requirements
- How we handle requirement changes
- Use case diagram

Requirements

User Stories Using PivotalTracker : Workflow

- Add ideas to Icebox
  - Include test cases
  - Include list of tasks to complete
- Assign (estimated) difficulty of completing each feature
  - Points: Range 0 to 3 (3 being most difficult)
- Move user stories to Backlog when it should be done for upcoming iteration
- Move user stories to Current when its' development begins
- Requirement Lead accepts each user story once it is verified that it is tested and functional

Requirements

Sample User Stories: Features and Bugs

Boss Reacts to Mistakes

As a user, I want to see the boss react when I fire an incorrect letter by changing attack patterns so that I am discouraged from making mistakes as I continue to play.

BUG #11:

Shoot letter that is not collected

If player selects a letter that he/she does not have in inventory, Albert will still shoot it at boss.

Tasks:

1. Add collision detection to boss to sense wrong letter
2. Add animation to boss for the reaction

Test Case:

1. Shoot a wrong letter at boss
2. Verify that boss attacks player

Mostly track bugs in **Github**.

Requirements

Functional Requirements

Sample:

The player will start a new game by selecting a difficulty level and choosing which spelling lesson he or she wants to test in. The system will keep track of the player's selections and apply attributes of the selections towards gameplay.

Requirements

Non-Functional Requirements

Sample:

Algorithm Fallibility (possibility of it failing)

As a user, I should not experience game algorithm fallibility regardless of the frame rate of the machine I run the game on.

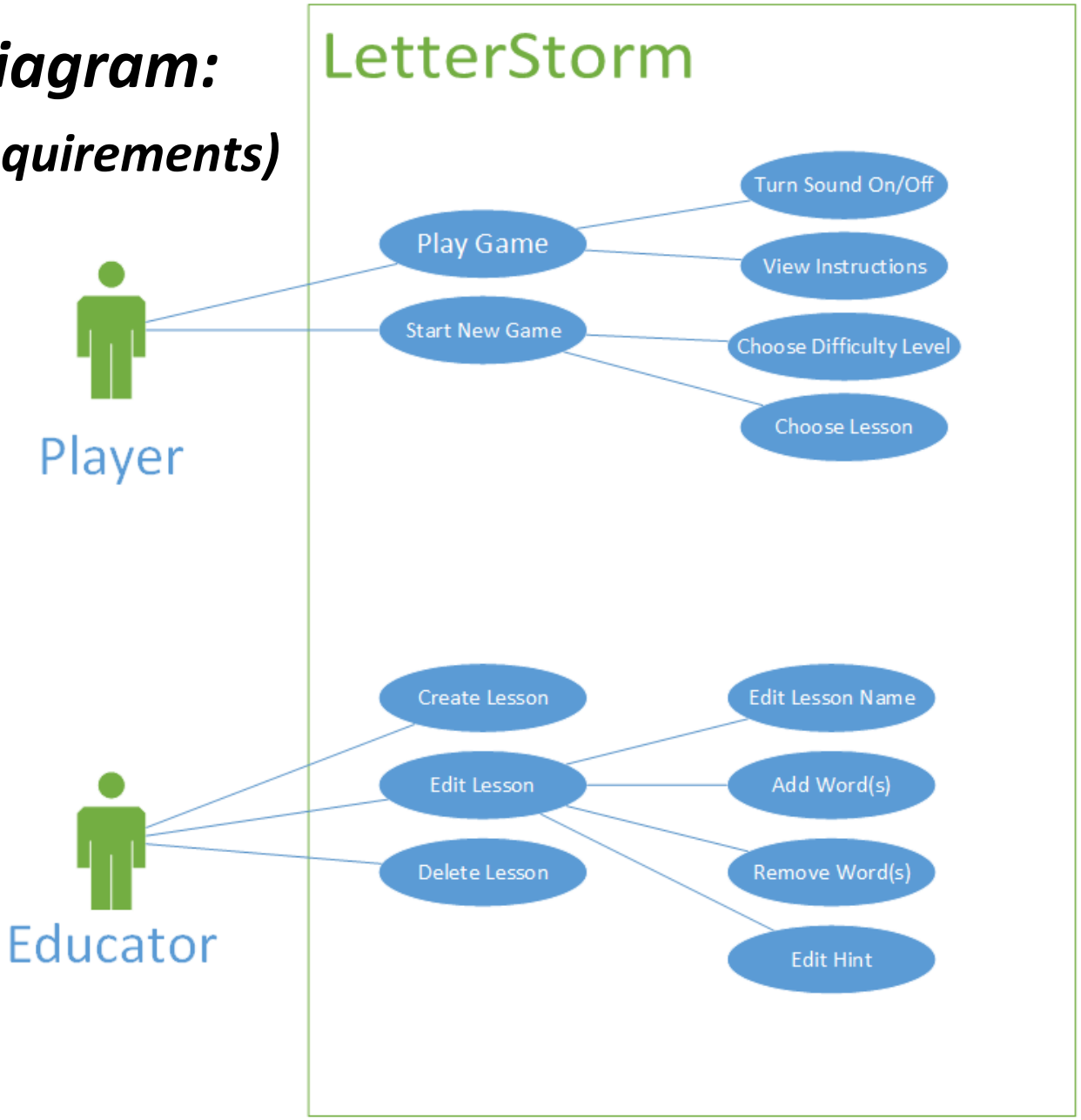
Requirements

How We Track Requirements & Changes

- Primary method: Weekly in-person meetings and daily emails
  - Go over all project requirements progress at start of each iteration
  - What's done and not done
  - What should be dropped or kept
- User stories belonging to requirements on Pivotal Tracker
  - Add user stories
  - Comment on updates
  - Comment or create user stories on discovered issues
- Bug Tracking
  - Some bugs listed in Pivotal Tracker, but official list of issues are on **GitHub**

Use case diagram:

(functional requirements)



Requirements

Iteration 3 Requirements

- Smart Enemy movement patterns (Done)
- Sound/Music (Done)
- Power-Ups (Done)
- Finalize and add look of HUD and menus (Done)
- Dropped features:
  - Selection of avatar
  - Save/Load game

Requirements

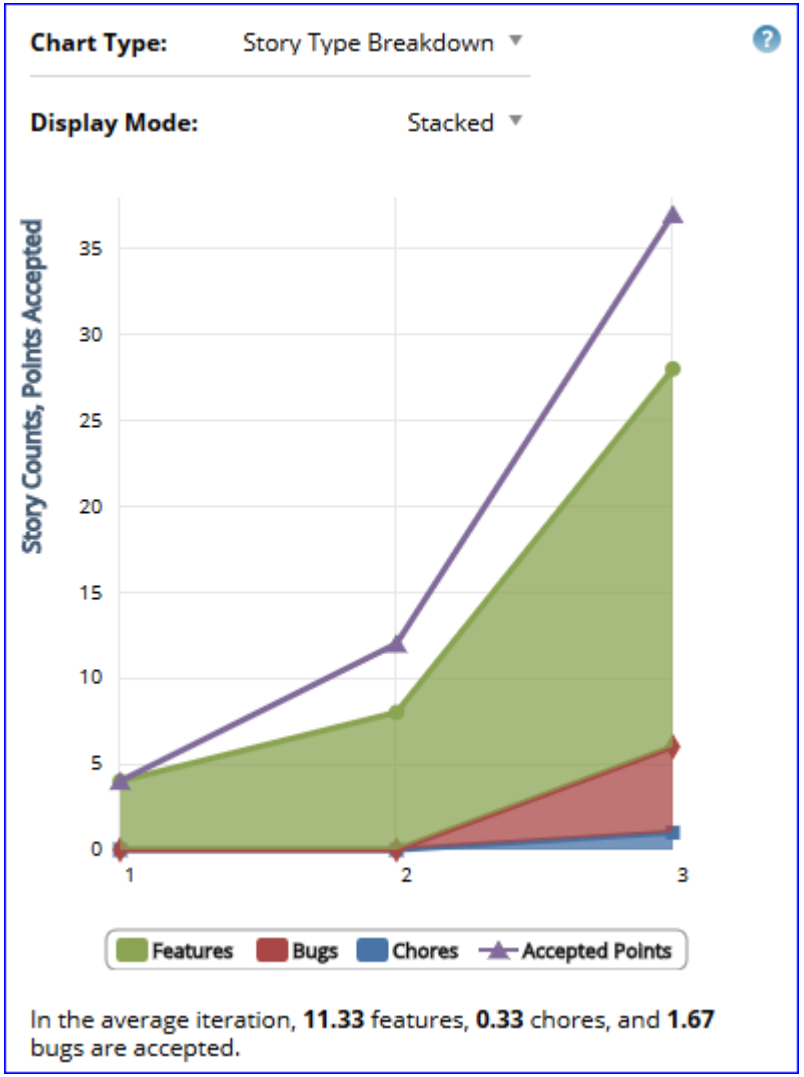
Requirements: Grand Scheme Checklist

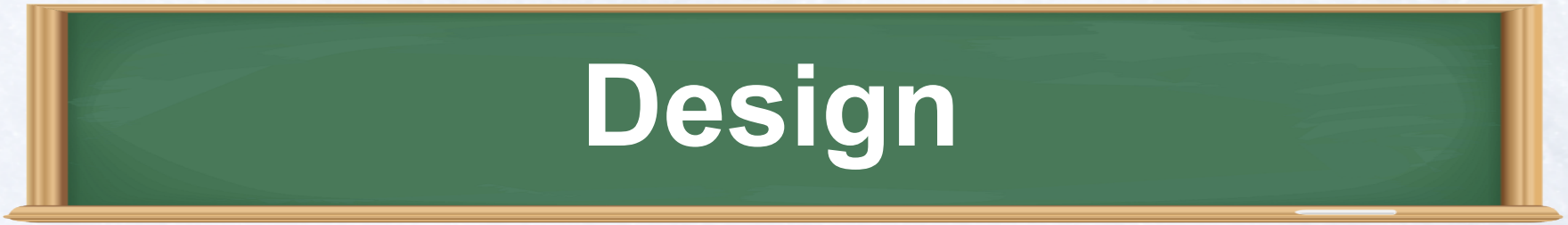
- General
  - The product must be a game that is playable and appropriate for children ages 5 and up
  - The game scenarios presented must enable the user to spell words (learning component)
- Target Environment
  - Product must be deployed as a standalone desktop application (Windows).
  - Optional: the product should be runnable on the web (html).
- Customizability
  - Game must provide an option to allow the user to add a custom word list or dictionary, which can be used as part of the game
  - Game must provide one or more default dictionaries that can be used as part of the game
  - Game must provide one or more characters for the user to choose from for gameplay
- Interface
  - A keyboard must be the primary method of input for using the software.
  - Optional: mouse or mouse-keyboard combination as input for the software.
  - A menu system must be provided for general navigation of the game
    - Main Menu
      - Play Game
        - ~~Select Avatar~~ Enemy Difficulty, and Word Dictionary
      - ~~Load Game~~
      - Manage Dictionaries
      - Settings [Moved to Pause Menu]
        - Sound and Music
      - Quit Game
    - Pause Menu
      - ~~Save Game~~
      - Return to Main Menu
      - Quit Game
  - A HUD must be displayed in-game to assist the user in keeping track of information required to play the game, such as health and letters collected
- ~~Functionality~~
  - ~~Game must be able to store the current state of the game in a save file~~
  - ~~Game must be able to load a game state from a save file~~
- Design and Strategy
  - Game must provide enemies for the player to avoid or defeat
    - Standard enemies
      - Passive enemies must drop consonants upon defeat
      - Active enemies must drop vowels upon defeat
    - Bosses must correspond to words that should be spelled
  - Game must provide collectible game items that will aid the user in beating the game
    - Power-Ups
    - Letters
  - Game must allow user to defeat standard enemies using one or more weapons
  - Game must allow user to defeat bosses using the letters acquired from all enemies defeated earlier in the level

Requirements

User Stories and Points

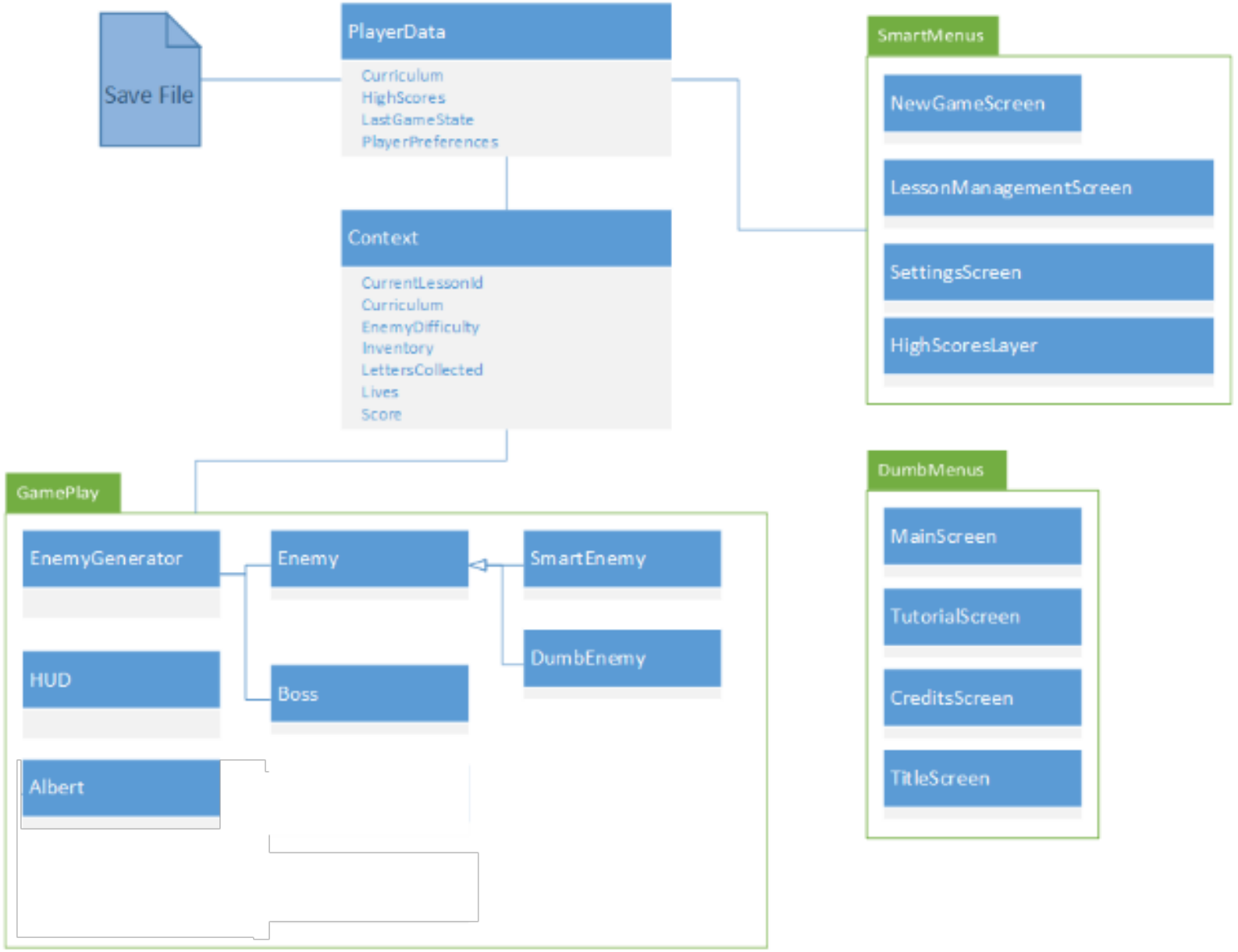
- Accepted:
  38/47 User Stories
  (including stories in the Ice Box)




Design

Component-based Development

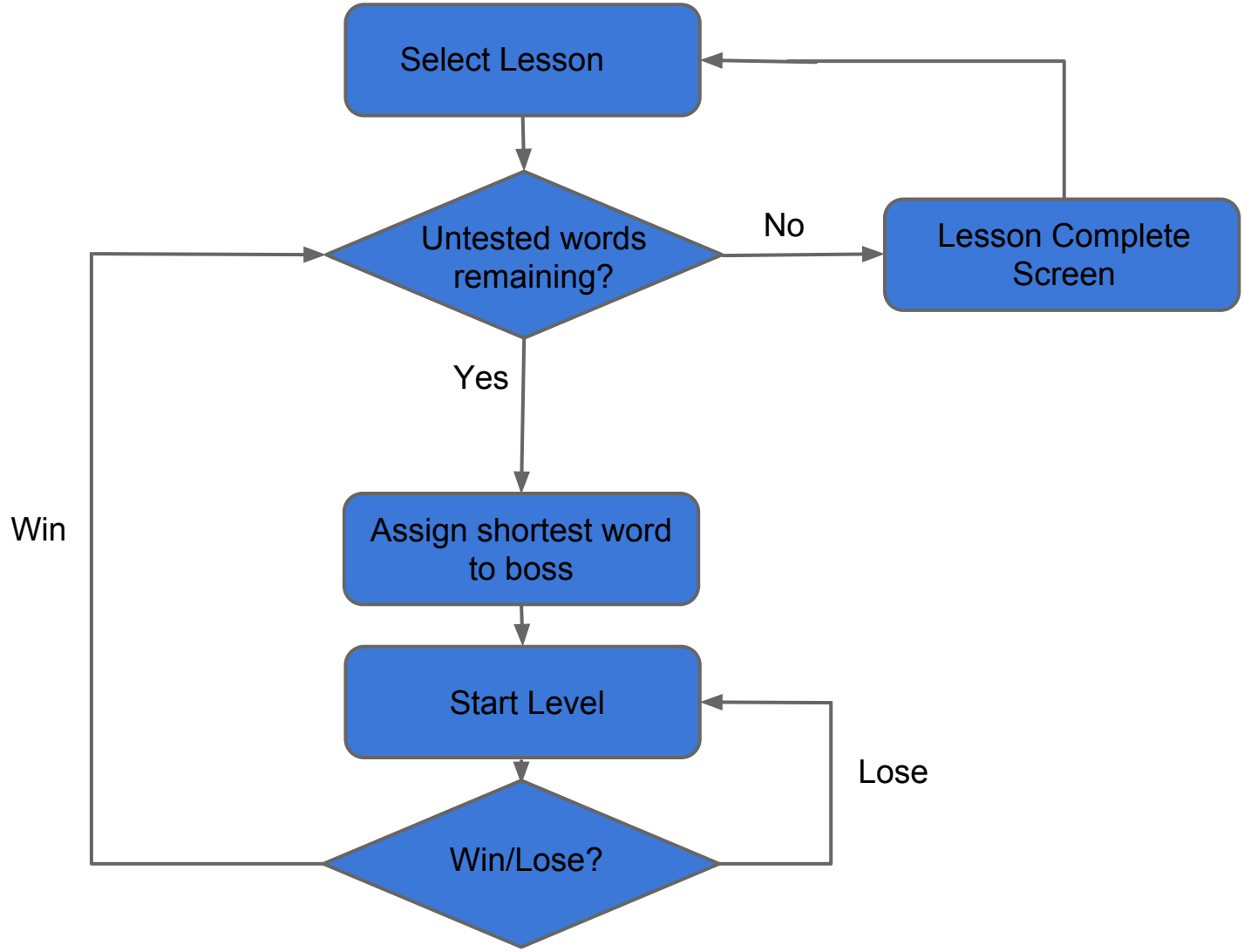
- **Player Data**
  - Saves/loads data, such as custom word sets (aka lessons)
- **Gameplay Objects**
  - Player, enemies and bosses, enemy generator, spawn points, HUD
- **Context**
  - Extracts certain player data and makes it available to gameplay objects as needed
  - Example: Enemy generator grabs word from context when level begins


Save File
PlayerData
Curriculum
HighScores
LastGameState
PlayerPreferences
Context
CurrentLessonId
Curriculum
EnemyDifficulty
Inventory
LettersCollected
Lives
Score
SmartMenus
NewGameScreen
LessonManagementScreen
SettingsScreen
HighScoresLayer
DumbMenus
MainScreen
TutorialScreen
CreditsScreen
TitleScreen
GamePlay
EnemyGenerator
Enemy
SmartEnemy
DumbEnemy
HUD
Boss
Albert

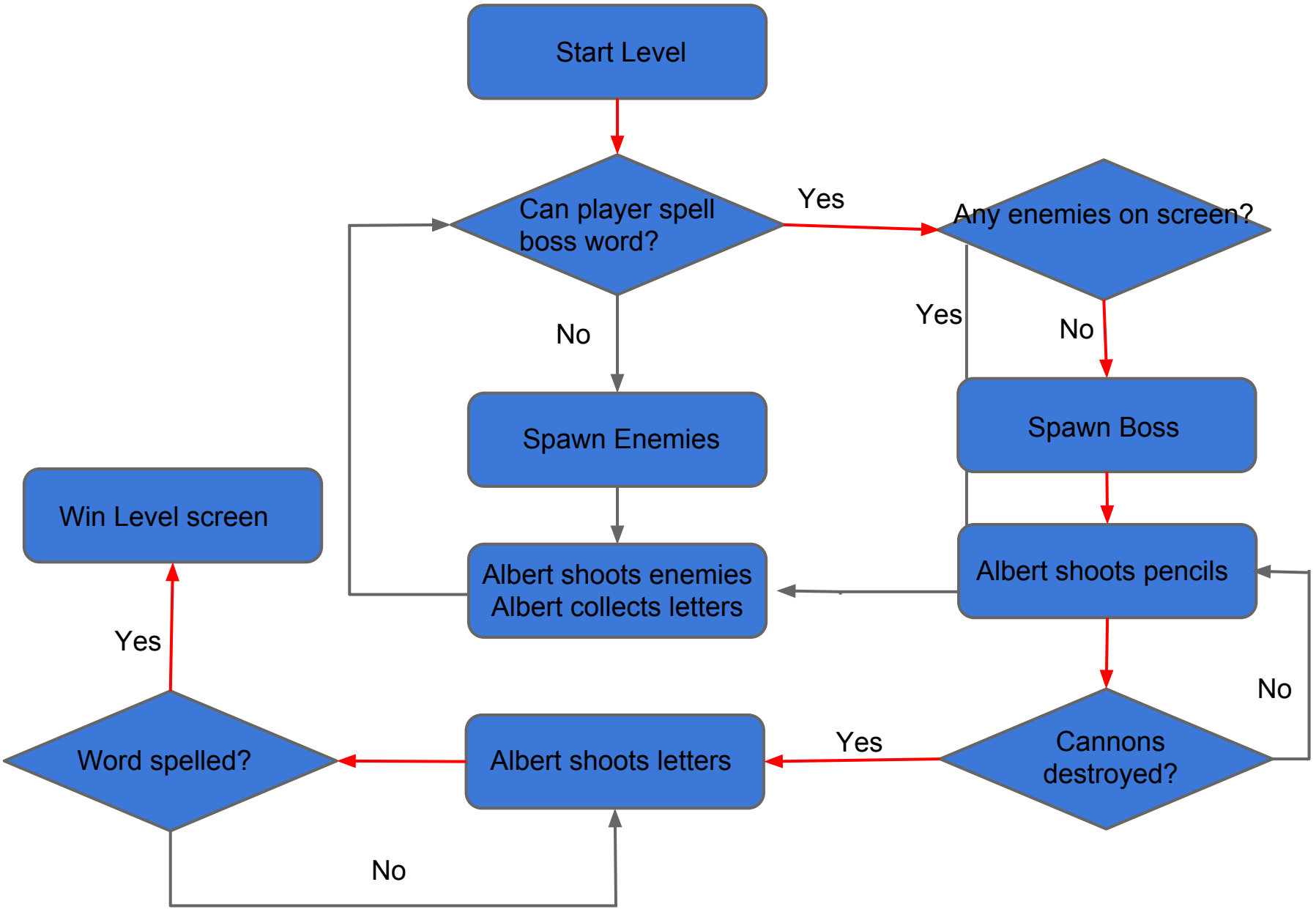
MVC...ish

- Rough correspondence:
  - Player Data ⟺ Model
  - Game Objects ⟺ View
  - Context ⟺ Controller
- Discrepancies
  - Business logic is really a combination of the EnemyGenerator (a game object) and Player Data
  - Player Data is directly manipulated by users
    - User ≠ Player

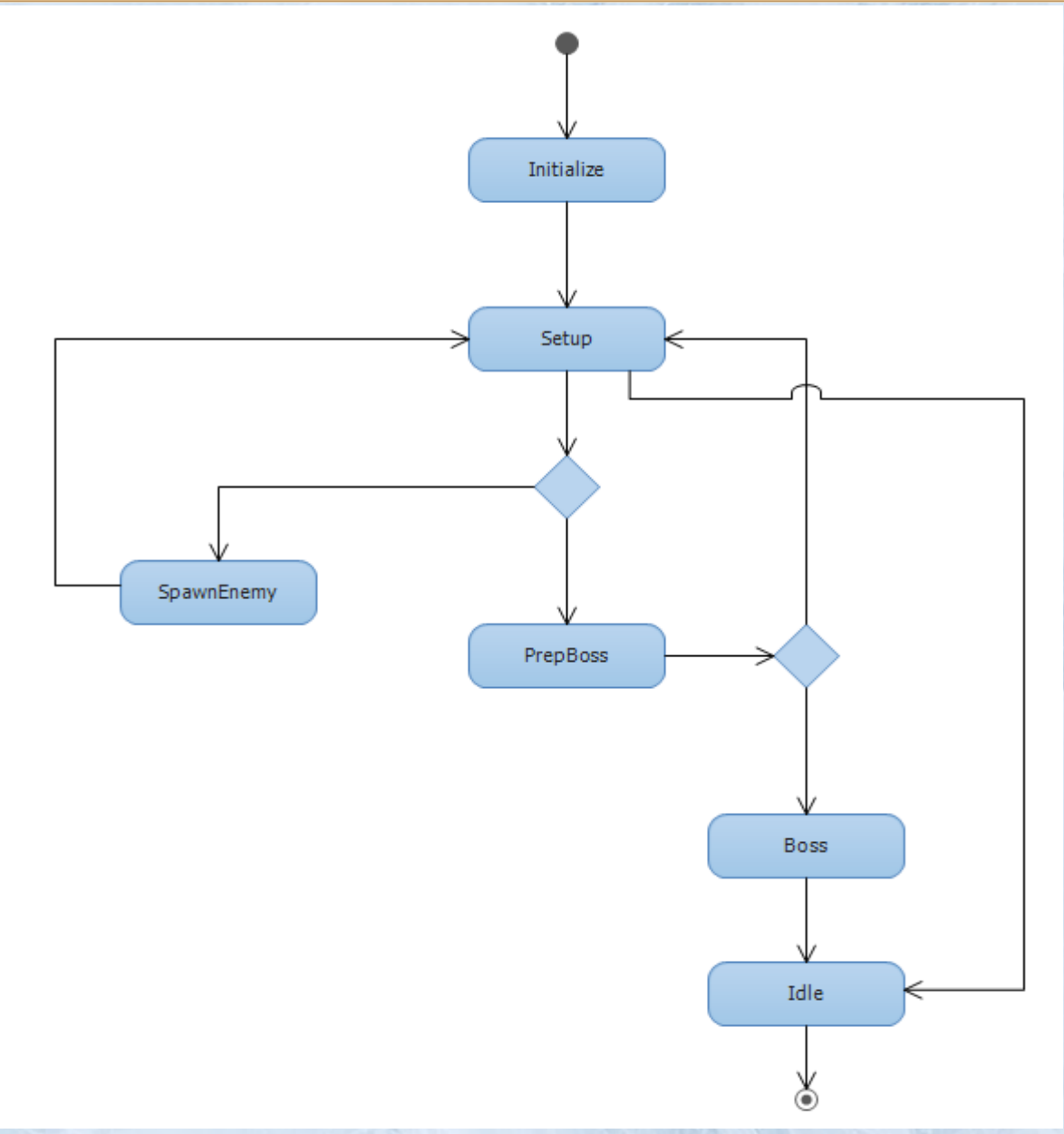
Lesson Flowchart



Gameplay Flowchart



Enemy Generator State Diagram



Enemy Generator Event System

- **Boss fight conditions**
  - Player has collected all letters necessary to spell boss word (Setup)
  - There are no enemies alive on-screen (PrepBoss)
- **Quality mechanics**
  - If the player can guess what the word will be because the only enemies are “D,” “O,” and “G,” that’s no fun (SpawnEnemy)
  - Limit the number of enemies on-screen simultaneously (SpawnEnemy)

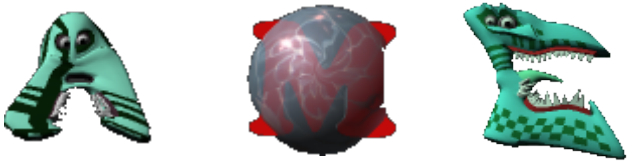
Collision Event System

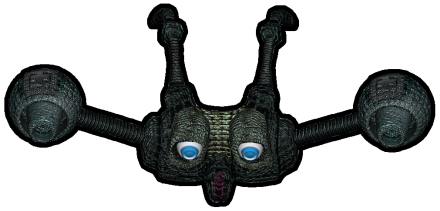
Other object: Boss, Enemy, Boss Projectile,Smart Enemy Projectile or Poisonous Mushroom
Result: Takes damage

Other object: Slowdown or Dual Pencil Powerup
Result: Gain corresponding power-up

Other object: Collectible Letter
Result: Corresponding letter added to inventory

Other object: Projectile
Result: Pain and suffering



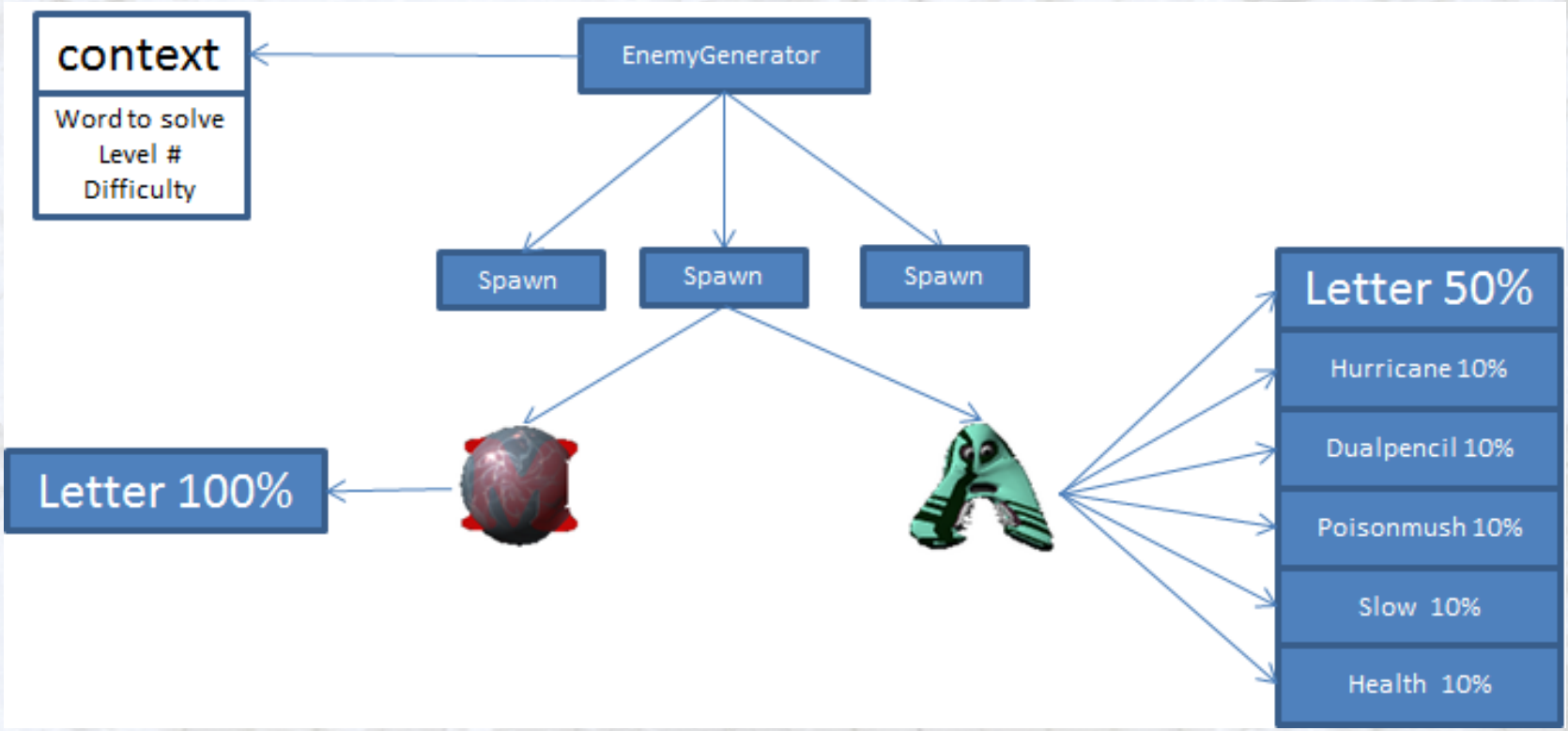


Other object: Incorrect letter
Result: Go into frenzy and charge Albert

Other object: Correct letter, word incomplete
Result: Display letter overhead

Other Object: Correct letter, word complete
Result: Proceed to “You Win” screen

Power-up and Letter Drop



Tools


unity


blender


GIMP


git


GitHub


PivotalTracker


Visual Studio


Google docs

Expertise (that contributed to the coding of the game)

- 3D Modelling - Nabil
- Game design - Nabil
- C# - David and Jeannie
- Code Documentation - David
- Git - Paul
- QA with background in C - James

Techniques

- Agile: Get it working > Clean it up/refactor > Add documentation > Enhance
- Pair programing: used to quickly integrate disparate code chucks
- Test-driven development: used for back-end data and configuration classes
- Informal group testing: coder notified of bugs found immediately following changes. Long-standing bugs are logged.

Coding Practices

Code has been formatted according to the .NET coding standard.

- **Namespaces**:
```
namespace GGProductions.
LetterStorm
```
- **Class names**:
```
public class ScoreKeeper
```
- **Class properties**:
```
public int Score
{
  get { return _score; }
}
```
- **Methods and arguments**:
```
public void Increase(PlayerAchievement achievement)
{ … }
```
- **Internally-used variables**:
```
private int _score = 0;
```
- **Enumerations**:
```
public enum PlayerAchievement
{
  DefeatSmartEnemy = 1,
  …
}
```

Coding Practices

- Namespaces
  - (Default Namespace) : used for classes that directly interact with a game object to reduce cross-class communication errors
  - GGProductions.LetterStorm.* : used for all data and configuration classes to promote structure

Integration & Testing Process

Fresh code base with continual testing

- **Same branch** (develop) is used for all work
  - Occasionally a feature branch was used for implementing features with large scope (ex: menus, enemy generator)
- **Push to Git** after adding new code and confirming that it does not break the product
  - Note: Code does not need to be fully functional to be pushed. This is to reduce the chance of merge conflicts.
- **Feature-specific testing** performed if new code is complete.
- **Informal group testing** (aka “checking it out”) performed by team upon receipt of feature-complete email.
- **Acceptance testing** performed by each team member during last week of iteration. This involves testing the product in depth and either addressing or logging any bugs found before merging to main.

Refactoring

As classes evolved, refactoring was used extensively and continuously to increase independence and encapsulation of functionality.

- Example:

Before	After

`public class Context {`
`  int decreaseHealthFactor;`
`  Health playerHealth;`
`  …`
`}`
`public class Health {`
`  …`
`}`
	`public class Context {`
	`  Health playerHealth;`
	`  …`
	`}`
	`public class Health {`
	`  int decreaseFactor;`
	`  …`
	`}`

Challenges

- Learning
  - game development (component-driven development)
  - the Unity framework
  - advanced C# concepts required to deliver a good game experience
  - Git
- Merge conflicts
- Keeping the broader scope in focus
- Keeping Nabil from overachieving

Implementation Examples

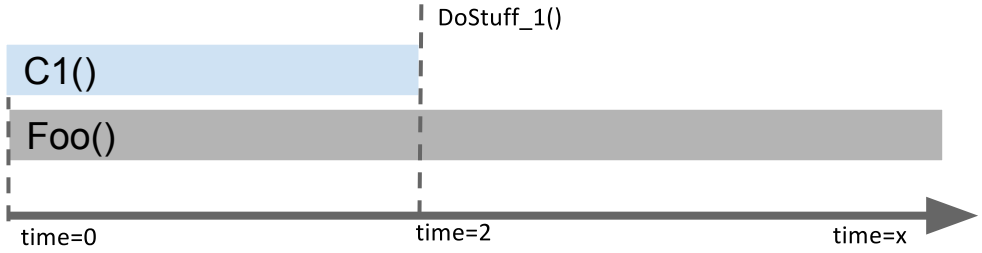
(aka coding the boss)

Coroutines and Stacking

```
void Foo (){doing_foo..}
void Update (){
Foo();
}
```


```
IEnumerator C1(){
Yield return new WaitForSeconds(2);
DoStuff_1();
}
```

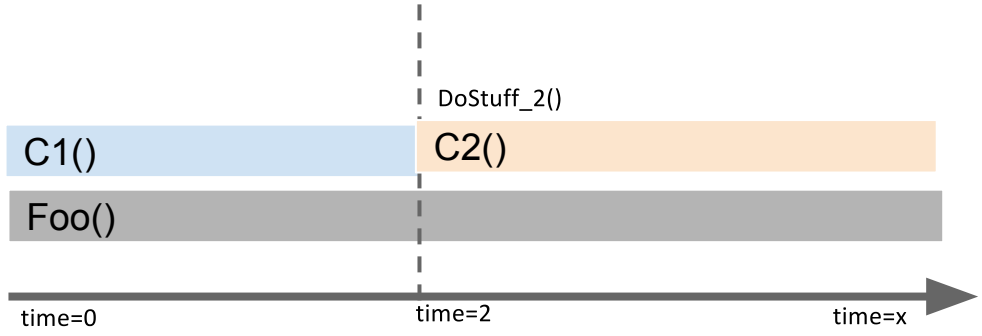
```
void Foo (){
StartCouroutine(C1)}
void Update (){
Foo();
}
```




```
IEnumerator C1(){
Yield return new WaitForSeconds(2);
}
```

```
IEnumerator C2(){
Yield return StartCoroutine(C1);
DoStuff_2();}
```

```
void Foo (){
StartCouroutine(C2)}
void Update (){
Foo();
}
```

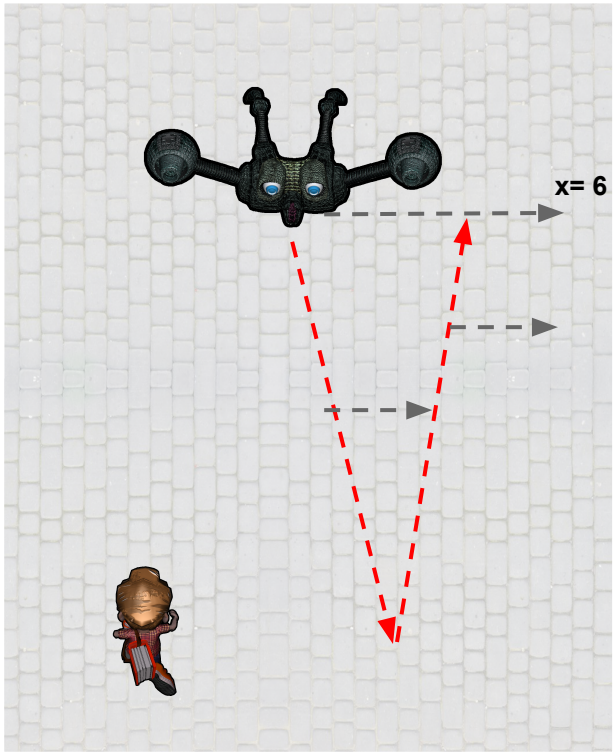


Boss Attack

Boss_Motion_animation.cs

```
void Update{
  float timeElapsed+=Time.deltatime;
  float factor = Mathf.Cos(timeElapsed);
  transform.Translate(Vector3.right * (factor / 20) *
Time.timeScale, Space.World);
if(switch)StartCoroutine("attack");
}
```

```
IEnumerator curle(){animation.CrossFade("curl");
yield return new WaitForSeconds(animation["curl"].length);
       }
IEnumerator downUp(){
while (transform.position.z < -4f){
            moveDown();
            yield return 0;
            }
while (transform.position.z > -4f){
            moveup();
            yield return 0;
            }
            yield return null;
      }
```

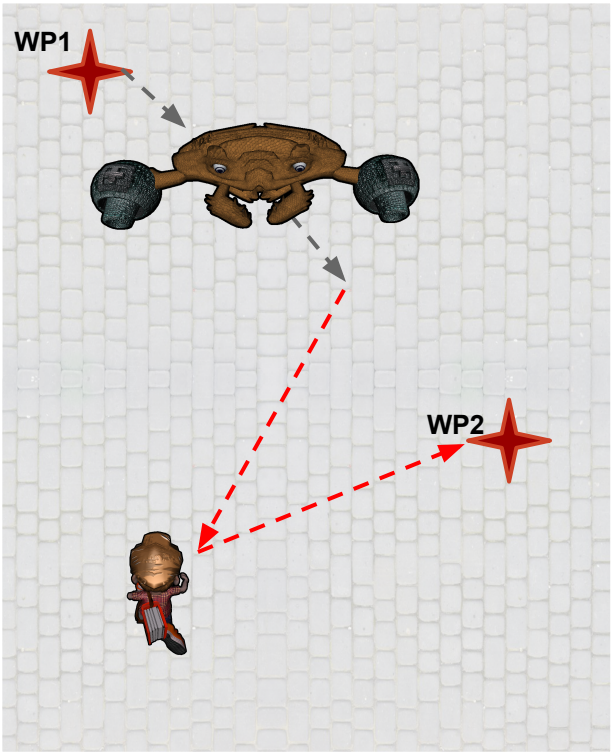


BigBoss Attack

BigBoss_Motion_animation.cs

```
private int Curr_waypoint;
void Start(){
  foreach (Transform c in waipointBlockPrefab.transform){
  transList.Add(c);
  }
}
IEnumerator Waypoint_routine(){
    while (true){
      if (stop_followingWaypoints){ break;}
      calcCurnext();
      yield return StartCoroutine(Move_A_B_time(transform, transList[CurIndex].position,
                                                transList[nextIndex].position,
                                                3.0f));
                                    CurIndex++;
    }
 yield return StartCoroutine(move_back_from_Albert_location(transform,transList[nextIndex].position, 0.3f));
}

IEnumerator move_back_from_Albert_location(vectors A, Vector3 B, 1.5f){
 yield return StartCoroutine(move_Toward_Albert_location(transform,Albert.position, 0.3f))
StartCoroutine(move_A_B(Albert.position, 0.3f),transList[nextIndex], 1.5f);
}
```



Testing and Metrics

(aka refining the monster)

Testing

- Manual Testing
- Individually after implementing code, either fixed on the spot or logged
- During weekly meetings
- Did not test latency
- Anything out of the ordinary

Bugs

- Used Pivotal Tracker
- logged all Bugs at the beginning of Iteration 3
- 11 Found
- 11 Resolved

Metrics

Iteration 1

- User Stories: 14
- Completed: 6
- Bugs Found: 0
- Bugs Resolved: 0
- Open Issues: 8
- Lines of code: 6644

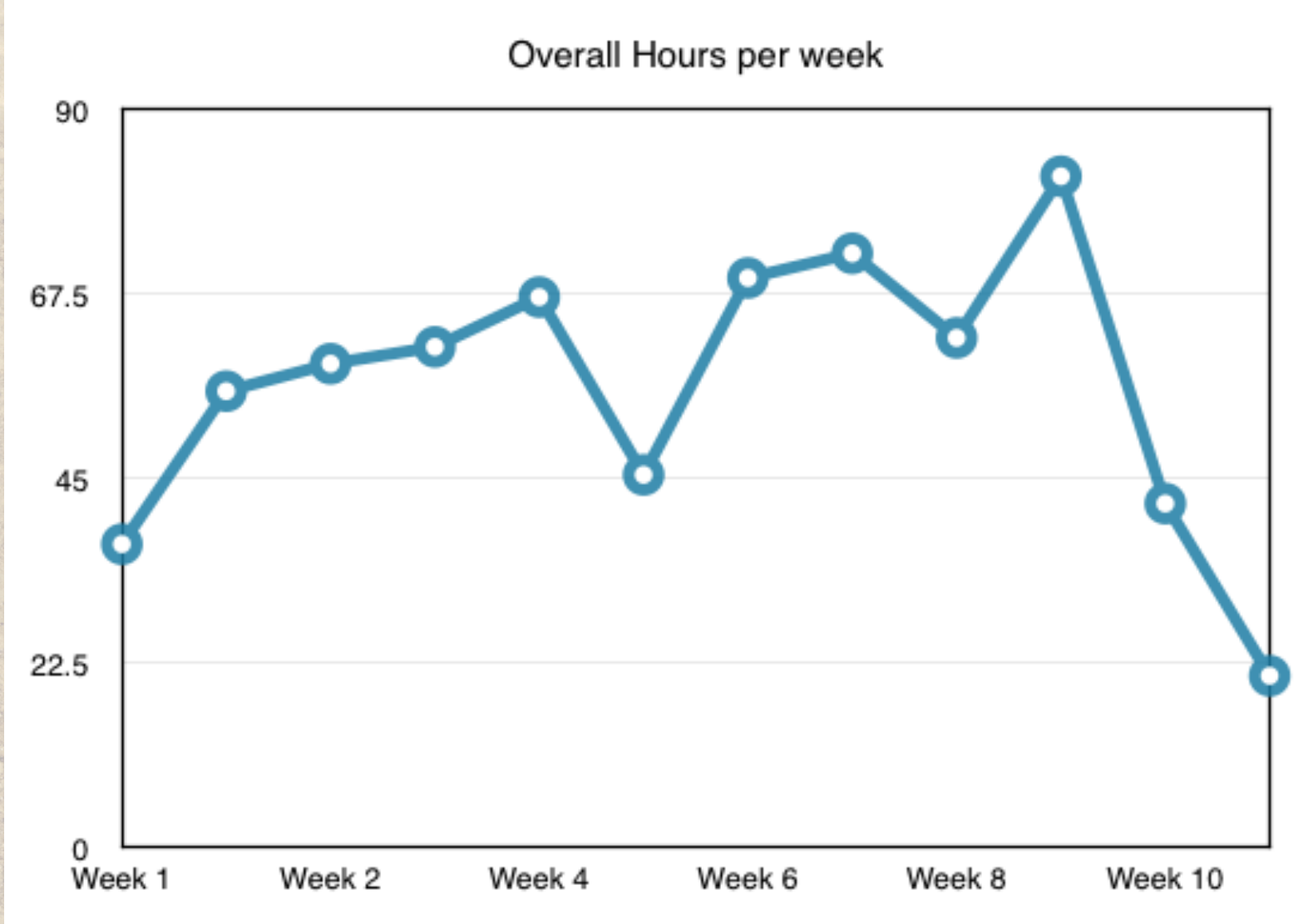
Iteration 2

- User Stories: 21
- Completed: 15
- Bugs Found: 0
- Bugs Resolved: 0
- Open Issues: 6
- Lines of code: 6450

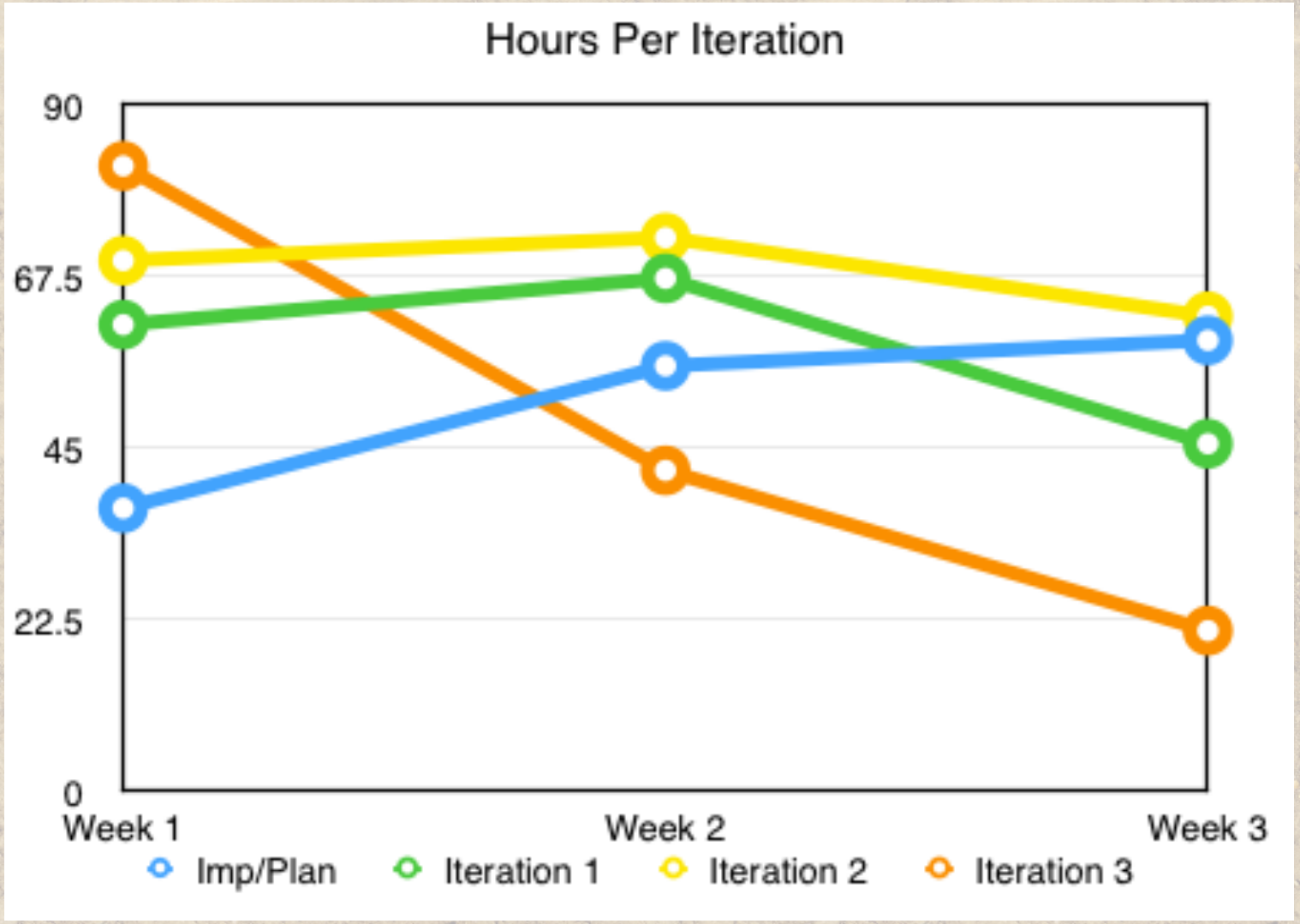
Iteration 3

- User Stories: 23
- Completed: 18
- Bugs Found: 11
- Bugs Resolved: 11
- Open Issues: 13
- Lines of code: 9842

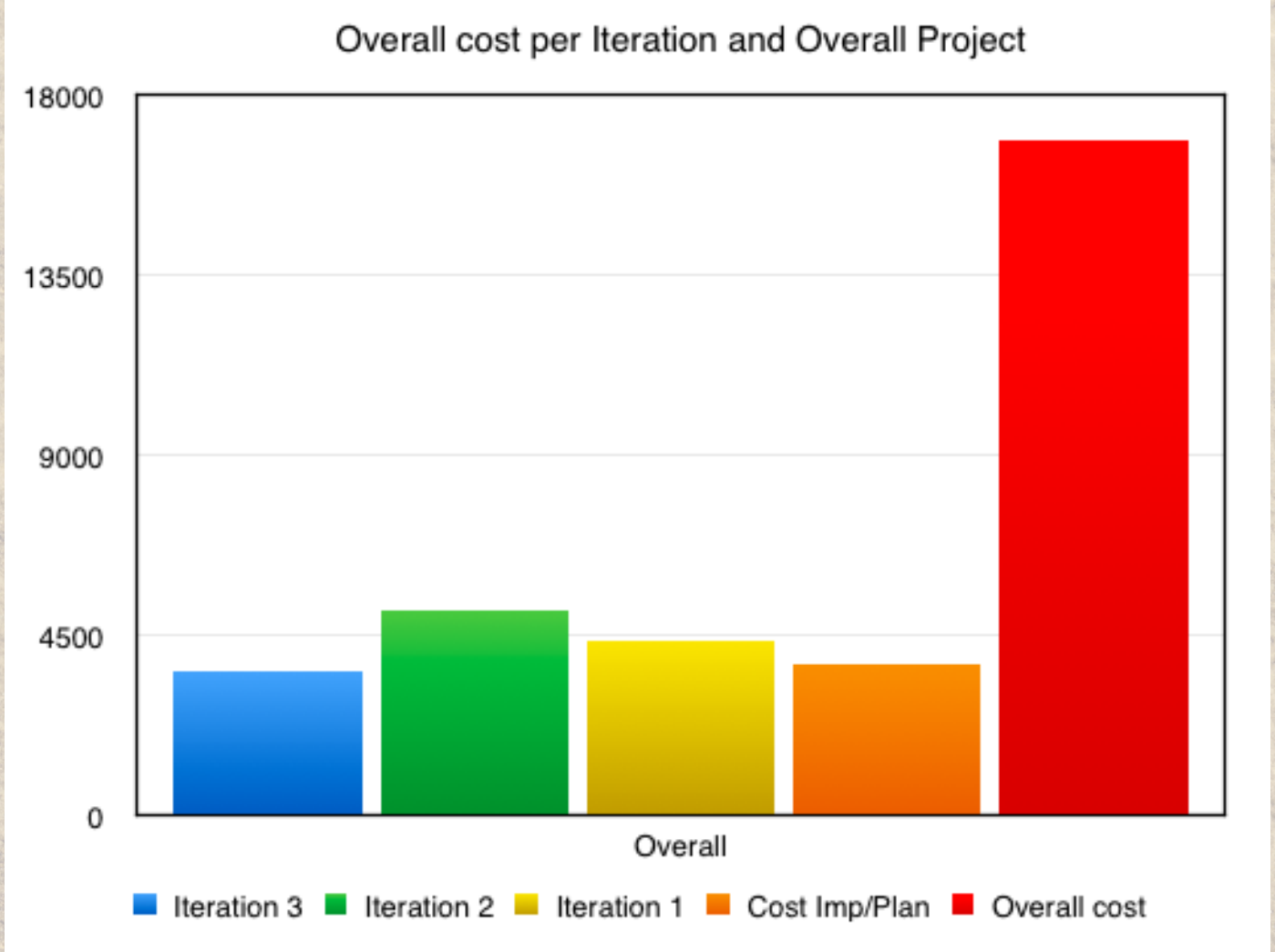
Metrics cont



Metrics cont



Metrics cont

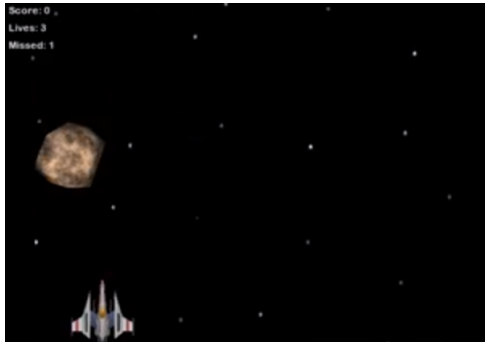


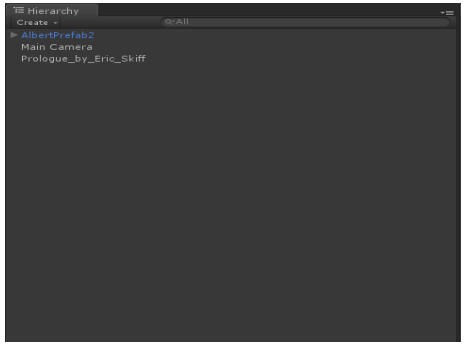
Project Management

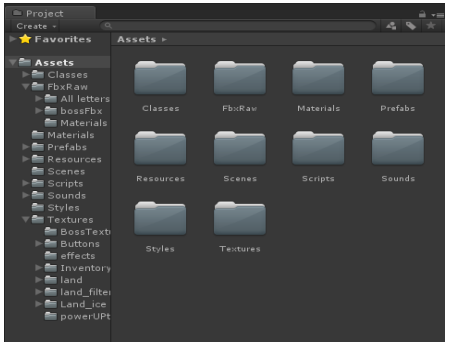
Learning and Brainstorming

-Getting the team acquainted with Unity3d:

Everyone followed a simple Unity tutorial to create a very basic 2d shooter

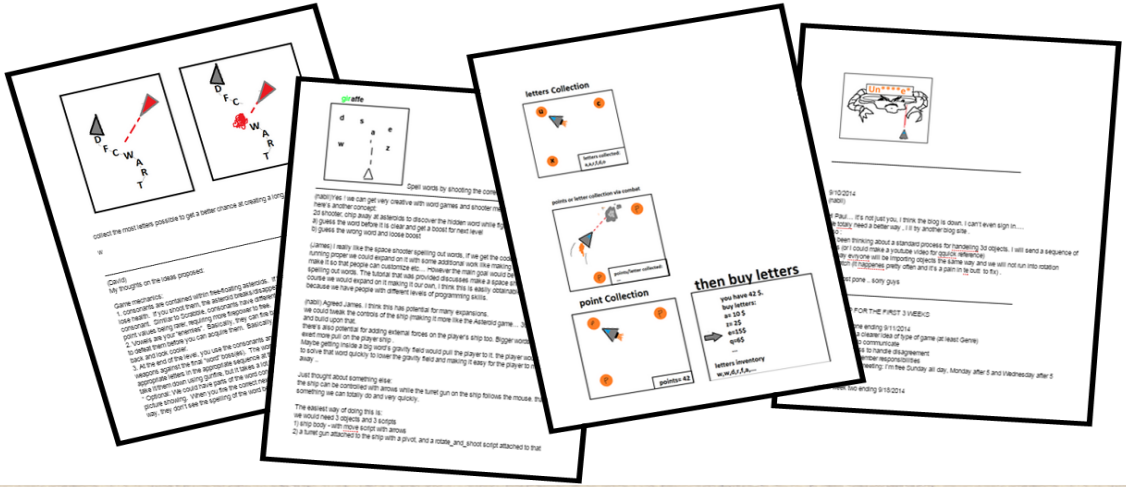






-Google Drive Brainstorm Doc:

Everyone contributed ideas, and interesting links with the team



Planning and Developing

- **Weekend Meeting**

  Face to face meeting

  Very casual

  Brainstorming ideas and problem solving

  Discuss next week's tasks

- **Daily Scrum email**

  Also very casual

  What was done yesterday

  What will be done today

  Potential blocks

- **Unscheduled meetings**

  Anytime is a good time

- **Coding teams**

  Team demo

  2 coder team

  3 coder team

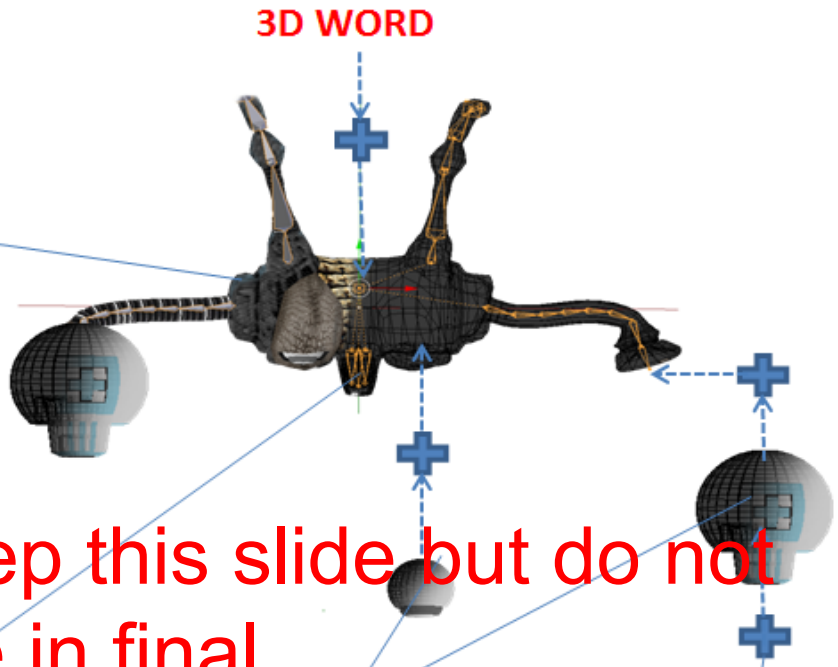
Fun Facts

- Don’t like what you see? LetterStorm is fully open-source, so you can make it better at https://github.com/GGProductions/LetterStorm
- Did you know that our project has it’s own marketing website? http://www.lifespen.com/
- And now, what you’ve all been waiting for…

Demo

Thank You!

Boss construction



Boss_3d_wordGen.cs

AllertAlbert()
"gun has been destroyed"

Construct_3d_word()

HandleCollision()

BoldFont = name of Object
Empty GameObject
A ← B B is child of A
Armature Bone
C# script
Script attached

Keep this slide but do not use in final

Boss_Motion_animation.cs

AnimationClip Action
AnimationClip Curlup
AnimationClip Uncurl

Boss_3d_wordgen.OnwrongLtr -> do Attack()

Update(){ Move(){}; }

Attack(){ Coroutine(); }

Coroutine()

Boss_eye_and_gun.cs

Find_albert()

Follow _Albert()

OnCollision() hitcount++

3dWordGen.AllertAlbertOnce( )

(note: collider is turned off)

BossShootInterval.cs

Gameobject bullet

Make bullet evry X second()

bullet

BossProjectile.cs

Move forward()
DestroyOutOfBound()
onCollision_Albert()

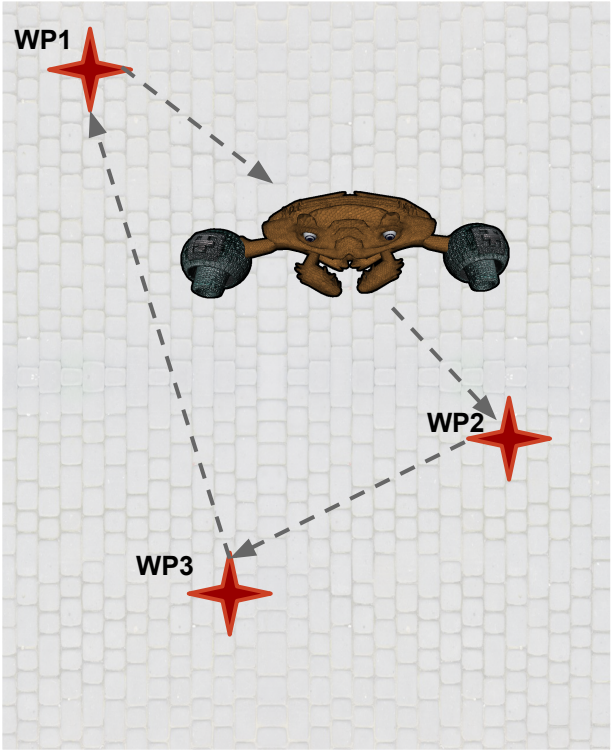
BigBoss Motion Script Components

BigBoss_Motion_animation.cs

```
private int Curr_waypoint;
void Start(){
  foreach (Transform c in waipointBlockPrefab.transform){
  transList.Add(c);
  }

}

IEnumerator Waypoint_routine(){
    while (true){
      if (stop_followingWaypoints){ break;}
      calcCurnext();
      yield return StartCoroutine(Move_A_B_time(transform, transList[CurIndex].position,
                                                transList[nextIndex].position,
                                                3.0f));
                                    CurIndex++;
    }
 yield return StartCoroutine(move_back_from_Albert_location(transform,transList[nextIndex].position, 0.3f));
}
```



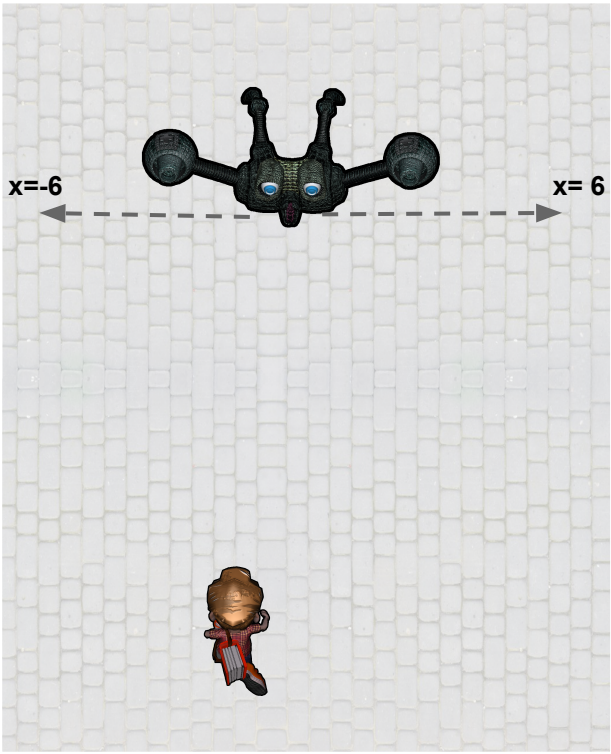
Keep this slide but do not use in final

Boss Motion Script Components

Boss_Motion_animation.cs

```
void Update{
  float timeElapsed+=Time.deltatime;
  float factor = Mathf.Cos(timeElapsed);
  transform.Translate(Vector3.right * (factor / 20) *
Time.timeScale, Space.World);

}
```



Keep this slide but do not use in final