# Level 3 EPQ
# *lssx*, LÖVE Space Shooter X

**ttxi**[*,1]
[*]JLC

**ABSTRACT** Developing a game requires a multitude of problem solving techniques to be in sync with a wide breadth of skills; ranging from geometry all the way to sound design. **lssx** is an attempt at creating a short yet re-playable experience that simulates the paranoia and anxiety from the Cold War by imposing upon the player an increasingly difficult challenge, to stay alive.

*"An archaic piece of software ripped from a fusty floppy disk is in-fact a US space-pilot training program from the Cold War."*

## Contents



Manuscript compiled: Tuesday 27th March, 2018
[1] ttxi | may.tt@protonmail.ch
Source for this project can be found at: https://github/com/twentytwoo/zephyr

## 1. Project Goals

Initial project were are to create a small, casual space-esque shoot-em-up with semi-realistic physics and retro graphics, akin to games of the 80's, a la. *Asteroids*.

- Create a *polished* game with plenty of replay-ability
- Publically release on itch.io and GitHub

## 2. Relevant Research

### *A. Game concepts*

Considering we know that the game genre should be space-related, we should look at the past to derive inspiration for *lssx*.

#### A.1 Cold War paranoia

The Cold War was a period of massive weapon stockpiling, societal paranoia, unprecendeted tension and more importantly, a fantastical imagination for the future of the human race, be it extinction by nuclear weapons or the development of an idea: that we could fare the stars and become a multi-planetary civilization.

It is often said that war drives the pace of technological advancement, the Cold War was no exception, in this a period of tension came huge development on nuclear weapons and ICMB's - advanced systems were created to oversee the deployment of WMD's in the case of all-out war. This lead to the development of computer systems such as the Mark 1, Whirlwind, ENIAC and others.

The Semi-Automatic Ground Environment system (SAGE) was one of these such computers, created deep inside caverns, these real-time computers tracked every movement in the sky in order to protect the USA from a sneak nuclear attack.[1]

**Figure 1** SAGE computer display[2]

The only relevant aspect of SAGE to a game is the aesthetics. The SAGE used a state-of-the-art console display to monitor Soviet bombers via the use of vector graphics projected onto a Situation Display (SD) via electron beam, similar to CRT's.

[1] I-PROGRAMMER. (2018). Sage - computer of the cold war, [Online]. Available: http://www.i-programmer.info/history/9-machines/441-age.html (visited on 03/27/2018).

### A.2 Space race

As part of *Operation Paperclip*, many prolific Nazi scientists were imported over from Germany into the United States - the purpose of this operation was to convince the scientists to simply work with the Americans on rocketry. By 1944 the Germans had mostly perfected their "*vengeance weapon*", the **V-2 Rocket**, or more affectionately named "*Retribution Weapon 2*" - a liquid-propellant powered rocket engine devised to strike British civilian hot-spots. Unfortunately for the Germans this weapon came too late in the war to have any real effect on the outcome of the war.

These scientists with a strong background in rocketry paved the way for the Space race, indeed the father of rocket technology and space sciences, *Wernher von Braun*, was imported from Nazi Germany along with 1,600 other specialists to develop a system (the *Saturn V* rocket) that would take the Americans to the Moon before the Soviets.


**Figure 2** US Space Race propaganda poster[3]

The victory of the Americans in the Space race lead to the eventual dissolution of the USSR, but also a new found interest into space exploration in the American people. Artists and musicians created art which expressed their hopes, dreams and ideas of what the future had to hold.

This concept was developed in **lssx**.

> An archaic piece of software ripped from a fusty floppy disk is in-fact a US space-pilot training program from the Cold War.

**lssx** makes small references to the Cold War and Space race in the form of assembler code scattered throughout the menu, game over screen and other sections.

For example:

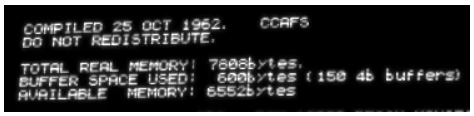



**Figure 3** Subtle nod to CCFAS

The height of the Cold War was during 1962, the same time as our fantasy game program was compiled. The code was also compiled at CCFAS, Cape Canaveral Air Force Station, the initial launch location of the Saturn V rocket.[4]

## *B. Similar titles*

The first step of developing a game is analyzing similar titles for inspiration, idea's and techniques. Studying these successful titles will also show what makes a game fun and enjoyable for players.

The following games will be analyzed:

- BYTEPATH
- Reassembly
- DATA WING

### B.1 BYTEPATH

Created by SSYGEN,

> BYTEPATH is a re-playable arcade shooter with a focus on build theory-crafting. Use a massive skill tree, many classes and ships to create your own builds and defeat an ever increasing amount of enemies.


**Figure 4** BYTEPATH gameplay

- Unique aesthetic
- Solid game play, focus on re-playability
- Repetitive yet enjoyable game loop
- Ability to customize ship

BYTEPATH has been developed with the LÖVE game framework, written in the Lua programming language. The game has also been released onto Steam, the main digital distribution platform for PC. SSYGEN has also made an effort to integrate Steam into BYTEPATH by the addition of Steam achievements. Achievements

[4] Wikipedia. (2018). List of cape canaveral and merritt island launch sites, [Online]. Available: https://en.wikipedia.org/wiki/List_of_Cape_Canaveral_and_Merritt_Island_launch_sites (visited on 03/27/2018).

are meta-goals defined outside the game's parameters, achievements add an element of progression that hooks the player into collecting all achievements and fulling completing the game.

BYTEPATH's gameplay is mostly centered around a unique play style wherein player attacks are stackable. The game is 2 dimensional. 2D games allow developers to be able to quickly develop game because of the simpler geometry math required (z-axis of movement removed), 2D games are also less resource intensive, hence their popularity on mobile devices.

### B.2 Reassembly

Reassembly is a spaceship building and universe-exploration game created by Anisoptera Games and released on the $19^{th}$ of February, 2015.[5]

> The main objective of the game is to collect resources, expand and grow your fleet, and conquer your personal galaxy, complete with stellar ambiance by the *Peaks*.

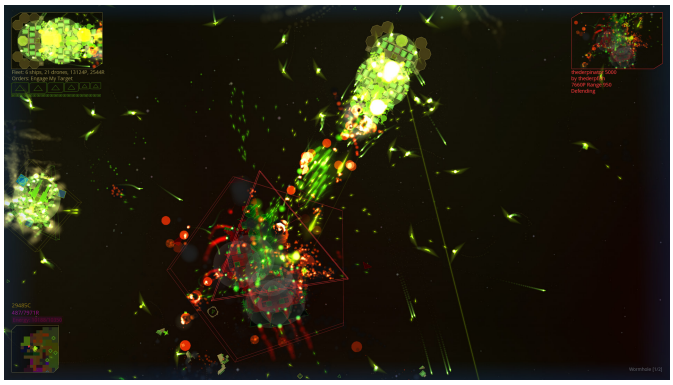


**Figure 5** Reassembly gameplay

- Almost endless replay-ability due to ship-crafting mechanics
- Beautiful use of GSGL shaders and lighting create a feeling of being in space
- Comfy soundtrack by the *Peaks*
- Massive spectrum of entity types

A large game play element of Reassembly is described in it's own name, players have the ability to create their own ships from a set of geometric shapes, each geometric shape has its own parameters that effect the performance of the ship as an entity. When shapes are damaged they break off from the main ship and a tractor beam reassembles the ship from a blueprint by collecting nearby transient parts.

Reassembly's creative, modular ship building has been compared to playing with Lego. The in game world is a single large open "galaxy", populated with rival factions. Players progress through the game by collecting resources, building a fleet, capturing territory, and activating damaged space stations.[6]

### B.3 DATA WING

DATA WING is a mobile game developed by Dan Vogt and released onto the Android operating system.

> Blast through a stylish, neon landscape in this story-driven, racing adventure. DATA WINGs deliver critical data throughout the computer system, following Mother's orders without question. But when the system comes under attack, and Mother becomes irrational, something must be done!

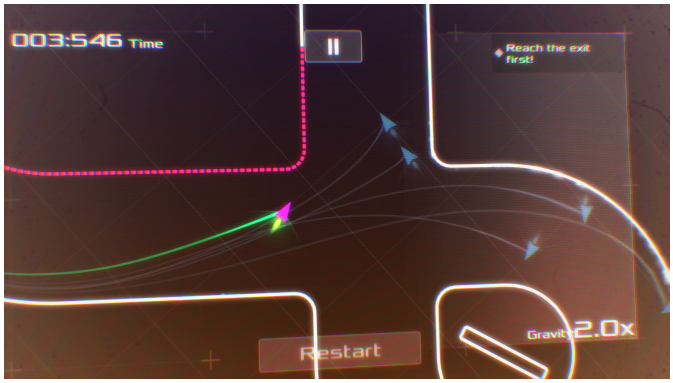



**Figure 6** DATA WING gameplay

- Simple movement mechanics
- Small scope results in very polished product
- Strong usage of camera parallax - not overused however

DATA WING's simplistic game play match perfectly with the mobile platform. Mobile players often prefer a casual experience which DATA WING is able to provide by mixing up racing and puzzle-solving, wrapped up in a bold narrative that packs an emotional clout.

One part of DATA WING that seemed especially important in shaping the game's aesthetic is the subtle use of camera parallax.

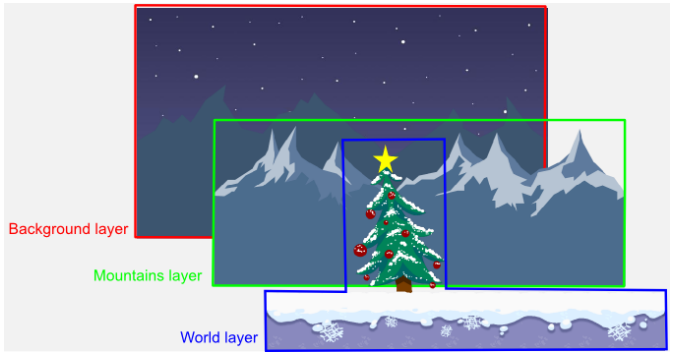



**Figure 7** Camera parallax

The parallax in DATA WING gives the user a sense of depth by moving objects at different velocities depending on their displacement from a virtual lens, shown in Figure 7. In most games parallax is achieved by separating groups of objects into *layers* which are moved independently of the player by some feedback.

### B.4 Conclusion

Each game has a different lesson to teach, a different style to show and all have displayed different and unique methods to achieve the most important aspect of any game, fun game-play. The main points gathered are:

[5] A. Games. (2017). Reassembly homepage, [Online]. Available: https://www.anisopteragames.com (visited on 12/05/2017).

[6] A. Games. (2017). Reassembly wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Reassembly_(video_game (visited on 03/26/2018).

- Emphasize on-screen particle / enemy count
  - High object count makes the game seem more interesting and fun to play as the player has more things to manage in mind, managing several aspects of game-play makes game less boring to play overall.
- Shaders
  - All three games have some form of shaders, from chromatic aberration to bloom which enhance the aesthetics of the game by adding interesting computer generated effects to the programmed graphics.
- Camera parallax
  - Reassembly and DATA WING use camera parallax heavily to make their UI and background effects more interesting and polished. Camera parallax also creates a feeling of motion when used in conjunction with player movement.
- First 30 seconds *must* be fun
  - Quoted from ex-BUNGIE developer Jaime Griesemer, "*if you don't nail that 30 seconds, you're not gonna have a great game*"

### C. Why LÖVE?

LÖVE (aka Love2D) is a free, lightweight gamedev platform developed by a vibrant community, enabling everyone to create 2D games relatively quickly.

Put simply, LÖVE is a blank canvas, where one is truly able to create whatever one wishes - no guidelines, design patterns or systems are enforced by the framework.

The programmer is able to create almost anything given enough effort and time via the incredibly powerful yet simple LÖVE Application Interface Program (API). LÖVE is mostly targeted at 2D games (hence why it's commonly referred to as "love2d"), however this freedom has allowed developers to simulate three dimensions via the use of ray-casting or by rendering millions of triangles with 3D math.

LÖVE is unique in the aspect that given all its simplicity, sophisticated projects can be created - the only limit with LÖVE is the developers imagination and skill.

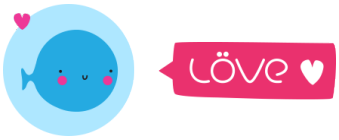



**Figure 8** LÖVE logo

### D. Architecture

A paramount component to writing any project is organization. If this key foundation is not laid out properly, a project has the tendency to spiral out of control in terms of complexity - this has the added effect of un-agile, un-flexible and often broken code. When projects are developed to such a scale such that the body of it cannot imagined in the mind, one relies on well written code to behave appropriately, regardless of state and IO.

In order to achieve this, several programming paradigms have been created in an attempt to encapsulate code into neat packages that know nothing of each other and deal only with streams of inputs and outputs.

This behaviour is known as "modularity". Modular code allows the programmer to develop applications in the form of many discreet modules all juxtaposed and working together in harmony. These modules can be swapped in and out, behaviour altered or removed without significant effect on the rest of the program as a whole.

#### D.1 Entity Component System

An *Entity Component System* (ECS) is an architectural pattern that follows values composition over inheritance. This allows for great flexibility when defining entities. Every entity consists of one or more components who's behaviours can be modified via the addition or removal of said components.

Structuring entities with components eliminates the ambiguity of wide inheritance hierarchies that is often the case with Object Oriented Programming. Common ECS approaches are highly compatible and often combined with data oriented design techniques.[7]

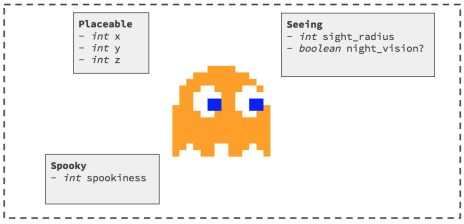



**Figure 9** Pacman ghost ECS

A major disadvantage to an ECS is the sheer amount of foundation required before any actual development can be begun. ECS is used at greater frequency in very large applications and is usually not suitable for smaller projects, Figure 10[8] explains.

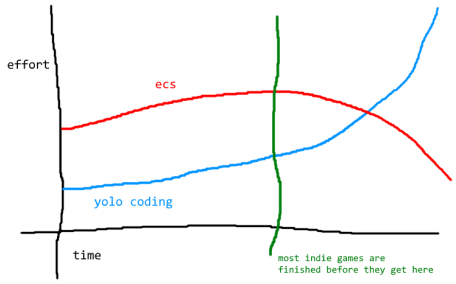



**Figure 10** Effort against time with regards to ECS

The time and effort required to set up a complete ECS is often wasted and unnecessary as the majority of single-developer projects do not reach the scale where the full power of an ECS can be fully utilized, thus a different approach should be taken.

#### D.2 Object Oriented Programming

*Object Orientation* (OO) is a programming paradigm that refers to a type of programming in which programmers define not only the data type of a data structure, but also the types of operations (functions/methods) that can be applied to the data structure.

[7] Wikipedia. (2018). Entity component system, [Online]. Available: https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system (visited on 03/06/2018).
[8] SSYGEN. (2018). Ecs vs yolo coding, [Online]. Available: https://github.com/SSYGEN/blog/issues/24 (visited on 03/06/2018).

The three cornerstones of object orientation are *Classes*, *Methods* and *Inheritance*.

Consider the following image, Figure 11.

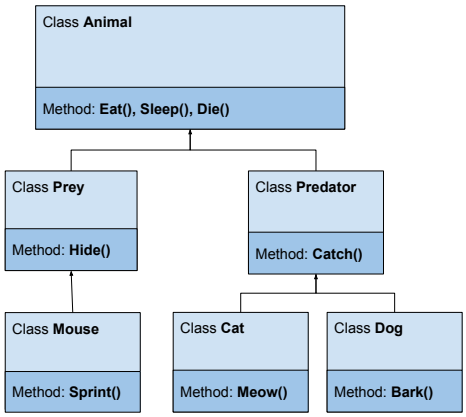



**Figure 11** A simple class structure

The class **Animal** is the *base* or *parent* class, and that **Prey** and **Predator** are *children*, similarly **Cat** and **Dog** are children of **Predator** and **Mouse** is a child of **Prey**.

Each class has its own *methods*, a method is an action in which an object, for example a cat, would take. In this example a **Cat** can `Catch()` a mouse, but it can also `Eat()`, `Sleep()` and `Die()`, children inherit their parents methods.

This way of constructing programs allows code to be abstracted into forms that are easily understood by relating objects into real world examples. An *instance* is a specific realization of any object, for example:

**Listing 1** MoonScript OO example

```
-- parent class Animal has x/y position
class Animal
  new: (@x, @y) =>

-- class Dog is and Animal
class Dog extends Animal
  new: (@age, @hungriness) =>
    super!
    print("Bark!")

  chew: (object) =>
    super\Eat(object)
    print("Mmmm...!")

-- Instances of object class Dog
Poodle = Dog(12, 0.1)
Beagle = Dog(2, 0.5)

-- Instance of class Bone
Tasty_snack = Bone!

-- Performing methods on instances
Poodle\chew(Tasty_snack) --> "Mmmm...!"
```

Objects can also have attributes, for example if a **Vehicle** was a parent class, **Cars** and **Motorbikes** would be typical children classes. An attribute of these classes would be the number of wheels, 4 for a **Car** and 2 for a **Motorbike**.

Ultimately Object Orientated programming provides a way to program polymorphism and encapsulation into a game, abstracting problems into simple, human-understandable concepts with an overall aim of reducing total mental load on the programmer.

### *E. MoonScript vs. Lua with respect to OO*

LÖVE by default uses Lua, Lua itself was designed as a scripting language and was not designed to be used as a language to be used to develop games. Due to this, it has no *native* OO capabilities (unlike other languages such as C++ or Java).

As a workaround, Lua later introduced the concepts of *metamethods* and *metatables* which allowed users to create their own "version" of OO. This method of doing OO results in verbose, ugly code which often times introduce bugs.

Luckily, there have been several attempts to make OO easier in Lua via the creation of custom **libraries**. A **library** is a collection of non-volatile resources used by computer programs, often to develop software. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications.

The most notable Lua OO libraries being *middleclass*, *classic* and *hump.class*. Unfortunately they do suffer from overhead because of the added abstraction (writing metamethods/tables directly is faster but more complex, it's a tradeoff.). **MoonScript** was designed to solve this problem.

**MoonScript** is a dynamic scripting language that compiles into Lua. It hands over the power of one of the fastest scripting languages combined with a rich set of features to the user. The precompilation process removes all the overhead from a OO library since the interpreter can hard-code values in a way a human could not without significant planning and time.

A series of tests were devised to judge which of these libraries was the fastest, or faster than MoonScript. The experiments tested the main features of OO, Objects, Methods and Inheritance.

#### E.1 Classes

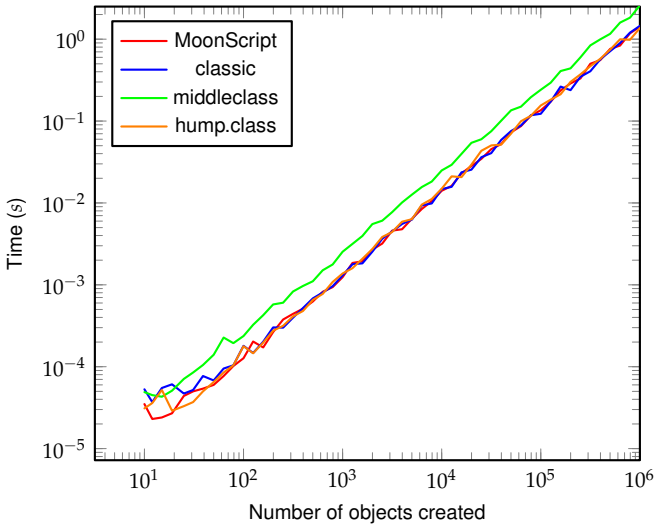
To test classes, a base class called `X` was created and inserted into a table `t` in ever-increasing quantities, from 100 to 10 million.[9]

**Listing 2** OO Test: 1

```
class X
  new: (@x, @y) =>

t = {}
for z=1, 6, 0.1 do
  for i=1, 10^z do
    t[#t+1] = X(10, 20)
```

[9] twentytwoo. (2017). Classes creation test code, [Online]. Available: https://gist.github.com/twentytwoo/38df41452b7ab047c316b0a8cdf34252 (visited on 12/06/2017).




The data shows that initially, MoonScript is the fastest at creating objects, and later all libraries become similar in speed when >1,000 objects created indicating a constant amount of memory is used when creating objects after the initialization. middleclass lags behind, being quite a lot slower than the others.

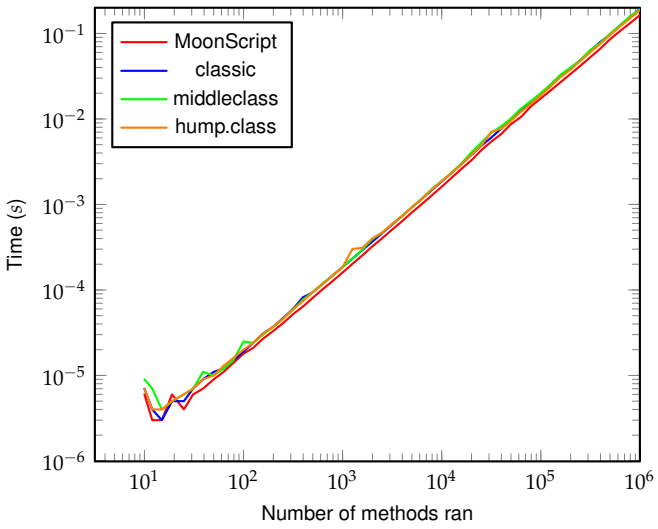
### E.2 Methods

Testing methods required the method `moveXtoY` to be added to class `X`, this method simply moves the attribute `X` to `Y`, as the name implies.[10]

**Listing 3** OO Test: 2

```
class X
  new: (@x, @y) =>
  moveXtoY: () =>
    @y = @x

for z=1, 6, 0.1 do
  for i=1, 10^z do
    t[i]\moveXtoY()
```




[10] twentytwoo. (2017). Methods test code, [Online]. Available: https://gist.github.com/twentytwoo/7f23960802416bf175fb557fe3ee9781 (visited on 12/06/2017).

As expected there is very little difference in the speed of execution amongst libraries since changing attributes is a very simple process and quite possibly simplified by the Lua compiler.

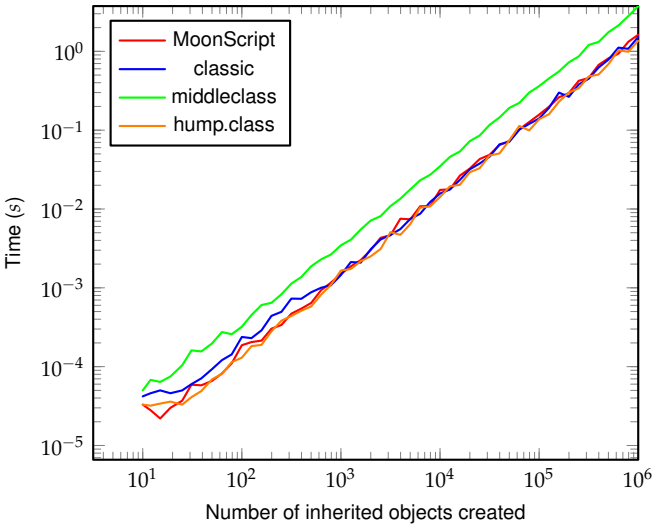
### E.3 Inheritance

To test inheritance, a class named `Y` was created which is the child of parent `X`.[11]

**Listing 4** OO Test: 3

```
class Y extends X
  new: (...) =>
    super(...)

t = {}
for z=1, 6, 0.1 do
  for i=1, 10^z do
    t[#t+1] = Y(10, 20)
```




middleclass again shows significant delays in creating objects and inheritance, MoonScript however is initially faster than all other libraries.

MoonScript chosen as it has the fastest OO, has greater readability because of the reduced syntactic sugar and therefore errors less likely to be made.

$$\text{MoonScript} \rightarrow \text{Lua} \Rightarrow \text{LOVE + Box2D}$$

The following MoonScript code:

**Listing 5** MoonScript examplar

```
Director.gameStart = () ->
  Timer.every 2, ->
    Pickup(math.random(2000), math.random(2000))
    Asteroid(100+math.random(1800), 100+math.random(1800))
```

Is compiled into the following Lua code:

Visually fewer lines of code reduces time taken to write code with the same functionality, it also reduces the chance of errors since syntax sugar is dramatically reduced.

[11] twentytwoo. (2017). Inheritance test code, [Online]. Available: https://gist.github.com/twentytwoo/75419cc33364571cd2cfc45d506c6d71 (visited on 12/06/2017).

**Listing 6** Compiled MoonScript code

```
Director.gameStart = function()
  return Timer.every(2, function()
    Pickup(math.random(2000), math.random(2000))
    return Asteroid(100 + math.random(1800), 100 +
     ↪  math.random(1800))
  end)
end
```

### *F. Box2D, the physics engine*

**Box2D** is an open source C++ engine for simulating rigid bodies in 2D.[12] It is widely considered to be difficult to use because of its complexity.[13]. In order to create realistic physics in my game I have to fully understand the workings of Box2D. The LÖVE framework has implemented a version of Box2D as closely as possible, this module is known as **love.physics**.

#### F.1 Worlds

All objects in Box2D exist in what is known as a "world", A world is simply an object that contains all bodies and joints, it also controls the global gravity and most importantly, set the collision callbacks for the world.

A collision callback is a function that is called when ever one of the four conditions is met, they are named aptly as:

- **beginContact**, Gets called when two fixtures begin to overlap.
- **endContact**, Gets called when two fixtures cease to overlap. This will also be called outside of a world update, when colliding objects are destroyed.
- **preSolve**, Gets called before a collision gets resolved.
- **postSolve**, Gets called after the collision has been resolved.

And can be set simply;

**Listing 7** Setting Box2D world callbacks

```
world = love.physics.newWorld(0, 0, true)
world\setCallbacks(beginContact, endContact, preSolve,
 ↪  postSolve)
```

There are a one or two caveats to these functions, the major one is that objects inside the world cannot be destroyed from inside the `beginContact` function. For example, if a bullet were to hit a player, you cannot destroy the bullet from inside `beginContact`, in order to overcome this problem I created a buffer. Commands can be added to this buffer and will be ran once the world update has ticked over. This means we can add instructions to destroy an object from inside the contact function, but they won't actually be run until a later time.

#### F.2 Bodies

Bodies can be thought of as a container of "stuff", a body has velocity, position and an angle. Separating the angle of the body from its contents allows us to manipulate the objects inside without having to deal with any awkward trigonometry.

This introduces another layer of complication, local and world coordinates. Local coordinates deal with the coordinates of objects within the body, world coordinates deals with the coordinates of the body on the LÖVE co-ordinate plane.

Inside a body you have things called fixtures and contacts.

[12] Box2D. (2017). A 2d physics engine for games, [Online]. Available: http://box2d.org (visited on 12/06/2017).

[13] as quoted perRude. (2017). Love.physics, [Online]. Available: https://love2d.org/wiki/love.physics (visited on 12/06/2017): *"love.physics is not lightweight, and not even remotely simple to use. It's a ten-ton hammer designed for heavy-lifting (er...hammer...lifting?"*

**Listing 8** General physics handling

```
Physics.update = (dt) ->
  lssx.world\update(dt)
  Physics.runBuffer()

Physics.addToBuffer = (func, hash) ->
    hash = hash or UUID()
    Physics.buffer[#Physics.buffer+1] = {func, hash}

Physics.runBuffer = () ->
  hash = {}
  if #Physics.buffer > 0 then
    for i = #Physics.buffer, 1, -1  do
      -- Detect if we've already seen this function before
      -- So we don't try and delete the same body twice
      if (not hash[Physics.buffer[i][2]]) then
        Physics.buffer[i][1]()
        hash[Physics.buffer[i][2]] = true
        table.remove(Physics.buffer, i)
```

#### F.3 Fixtures

A fixture a layer of abstraction between bodies and their shape, a fixture has no shape, yet it has density, restitution and friction, a fixture also deals with collision masks, group index's and categories which describe what fixture's shapes should have collision resolution with what.

A fixture is used to attach shapes to bodies, fixtures cannot be cloned, unlike shapes.

#### F.4 Shapes

Shapes are solid 2D geometrical objects which have form and control mass and deal with the actual collision *resolution* between other shapes. Shapes are attached to a Body via a Fixture. The Shape object is copied when this happens. The Shape's position is relative to the position of the Body it has been attached to.

There are five different types of shapes:

- **ChainShape**, A ChainShape consists of multiple line segments. It can be used to create the boundaries of terrain. The shape does not have volume and can only collide with PolygonShape and CircleShape.
- **CircleShape**, Circle extends Shape and adds a radius and a local position.
- **EdgeShape**,
- **PolygonShape**, A PolygonShape is a convex polygon with up to 8 vertices. Concave polygons can be created with multiple shapes and fixture in one body.
- **RectangleShape**, Shorthand for creating rectangular PolygonShapes.

#### F.5 Joints

Joints are used to attach multiple bodies together to interact in unique ways.

#### F.6 Contacts

The world has a "contact list", which lists all the collisions between objects Axis-Aligned Bounding Boxes (AABB's), this list can be

looked through to determine what is colliding with what, albeit somewhat inefficiently, as such they won't be used in this project much.

Figure 12 shows the relationship of love.physics components in a simpler fashion.

## 3. Development

### *A. Rapid prototyping*

To find out if such a project requiring complex physics simulation was possible, a principle of rapid development was followed, wherein one creates a basic game very quickly to prove that a concept is possible. The code is not meant to be smart or scalable. Satisfied with the results of the prototype, one can carry on and create a more intelligent design.

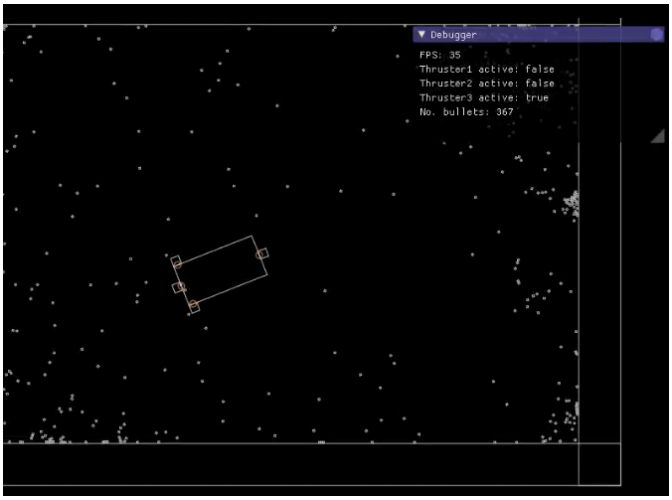



**Figure 13** A rapidly prototyped Box2D spaceship simulator

Initially the concept of destructible ships was explored by the addition of an algorithm that would detect nearby *structs* from other bodies, and copy all the shapes and fixtures from that body into our own.

*Structs* detect nearby *structs* and attach to each other (depending on which fixture is attached to the command module) if some state is on, if this state is off, then all unused structs are set to sleep (not detecting other fixtures)

*Structs* have no collision resolution because they are set to sensors - they let the game know something collided with them such that some specific function can be executed.

Shapes connected to a **CommandModule** (CM) will search for other *structs* within a given radius, if any *structs* are detected, the `beginContact` Box2D callback is triggered, an algorithm will run through the body that is not connected to the CM and clone all fixtures to the CM body and translate all vertices's of the fixtures to the *struct*.

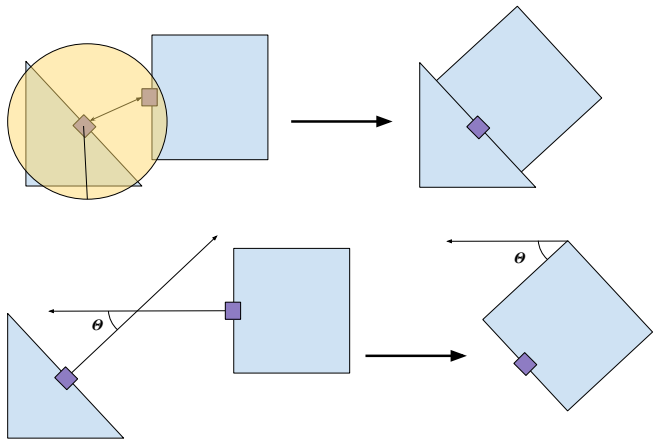



**Figure 14** *Struct* detection and other shape translation

Fixtures are only attached via *structs*, *structs* are specific positions on the shape where other fixtures can attach to the fixture.

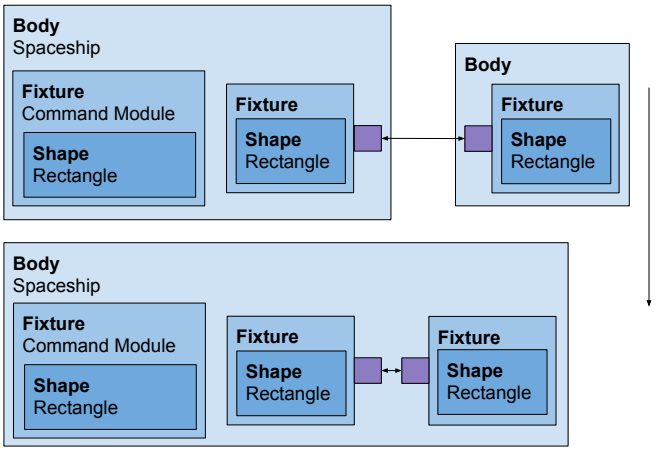



**Figure 15** The process of cloning other bodies contents into self

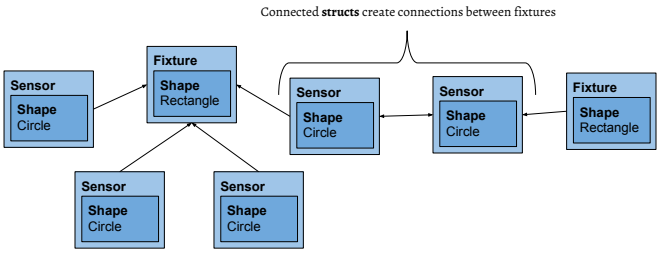



**Figure 16** Structs

- Detect which fixture is connected to the **CM**
- Clone all the fixtures from the body that isn't connected, use a transformation matrix to translate the shapes vertices to the **CM** *Struct* position.
- Destroy the other body

The following shows the most basic example of the algorithm, wherein the CommandModule (smaller shape) connects with a Square.

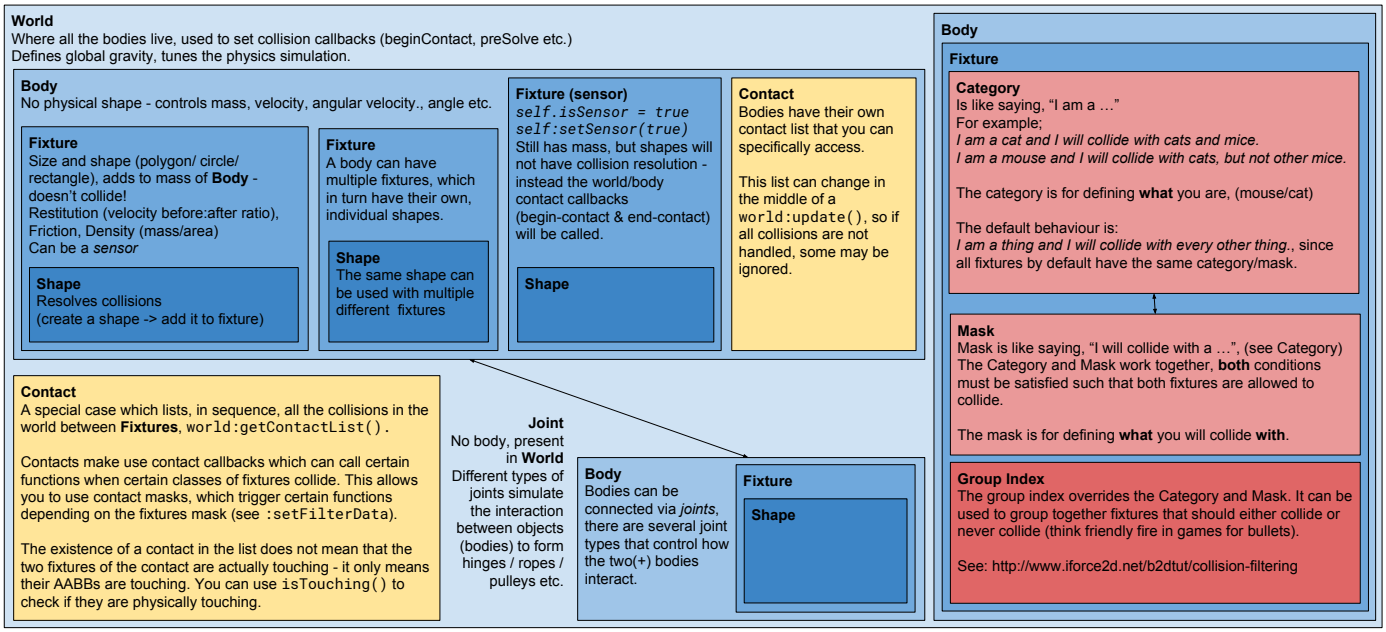



**Figure 12** Box2D explained

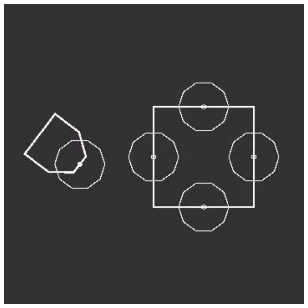


**Figure 17** Before *struct* contact

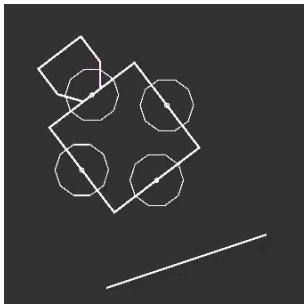


**Figure 18** Bodies cloned and joined together

This can then be scaled into larger, more complex shapes, shown in Figure 21.

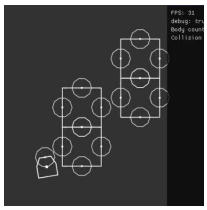


**Figure 19**

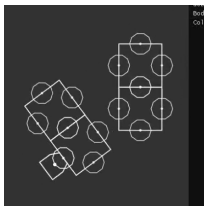


**Figure 20**

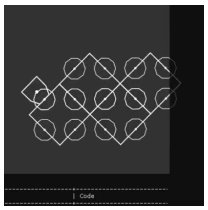


**Figure 21**

**Figure 22** Process of moving multiple shapes into one body

After a considerable amount of time, the algorithm for connecting shapes was perfected, however, this did not also deal with disconnecting shape - after consideration I decided that this direction would not be in the best interest of **Issx** because of the sheer time taken to perform a relatively small section of the game. **Issx** would be simplified to single fixture bodies only.

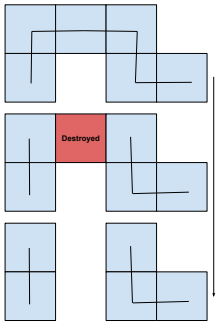



**Figure 23** Flood fill with connected shapes

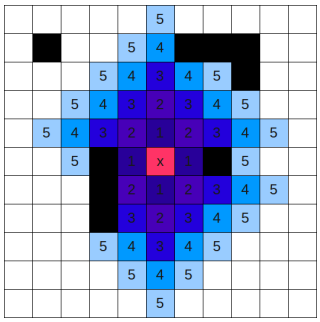


**Figure 24** Process of flood filling to see which shapes are still connected

After all this time I decided that this path was going to too large in scope and would create significant blockades to progress further down the line with respect to AI. So far I'd only managed to connect other components together, later on I'd have to write another algorithm to detach shapes when damaged.

### *B. Procedurally generated content*

Procedural generation is a method of creating data algorithmically as opposed to manually. In computer graphics, it is used to automatically create large amounts of content in a game.

In order to fill a world with interesting an unique content (not created by hand), we can make use of procedural generation and pass some random integers to a function and generate some output, the simplest example of this is the background, wherein several hundred small circles ranging in diameters are scattered about the map to give the impression of stars.

**Listing 9** Old PG background code

```
Background = {}

Background.load = () ->
  -- Create a static container for positions, this means
   ↪  they're only randomized once.
  export pos = {}
  sw, sh = love.graphics.getWidth(),
   ↪  love.graphics.getHeight()
  -- Insert 200 instances
  for i=1, 200 do
    table.insert(pos, {x: math.random(sw), y:
     ↪  math.random(sh), r: math.random(5)})

Background.draw = () ->
  love.graphics.setColor(60, 60, 60)
  for i=1, #pos do
    love.graphics.circle("line", pos[i].x, pos[i].y,
     ↪  pos[i].r)

return Background
```

### B.1 Asteroids

This process of using `math.random()` to generate unique content is repeated for asteroids, generating random vertices's of shape. Using procedural generation allows one to create a lot of content quickly. PG also makes the game look like it has more content than it actually does.

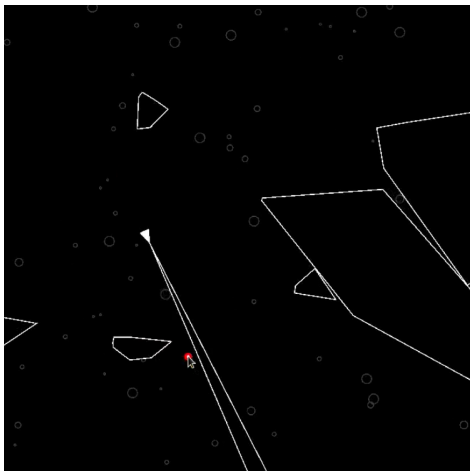


**Figure 25** Procedurally generated asteroids

Asteroids are destructible, when they take enough damage they crumble into smaller pieces that are ejected at some random velocity from the local center. The size of the parent asteroid dictates the size of the child asteroids in a realistic manner.

### B.2 Particles

Particles are used to great effect in almost all games, the best particles are those that aren't noticed by the player yet still add to the games' atmosphere.

**lssx** has a simple particle manager that procedurally generates and manages particles in the world, describing how long each particle should live for and how to should behave in the world. Due the the nature of particles, a large quantity are required, this means that they must be programmed as efficiently as possible in order to reduce memory usage and frame-rate spikes in the game.

The particles in **lssx** do not have a collision response with the world objects because simulating over one thousand collisions realistically would place a large amount of strain in the CPU, calculating positions, normals and forces that would need to be applied.

To this end, particles in **lssx** have a simple; x, y, dx, dy, wherein the particle will move to dx,dy position from x, y over a period of time. This involves the usage of linear interpolation, also known as *tweening*.

### C. Linear Interpolation and tweening

In order to move an object from one place to another over a period of time without instantly teleporting the object to the desired place uses a process called *Linear Interpolation* (lerping), commonly referred to as *tweening*.

Tweening is simply the process of manipulating an objects properties over time by a given function. For example, if a car has a constant acceleration, we can use a set of equations to calculate how long it would take such a car to move a specific distance, the same rule apply for inbetweening, whether if be changing and objects x,y position, health or even the amount of enemies spawned over time.

Luckily, the LOVE2D community have created a whole host of tweening libraries that can be used with a simple API. The one most commonly used is *flux*[14], created by **rxi**. **flux** is described as a fast, lightweight tweening library for Lua.

Any number of numerical values in a table can be tweened simultaneously. Tweens are started by using the `flux.to()` function. This function requires 3 arguments:

- `obj` The object which contains the variables to tween
- `time` The amount of time the tween should take to complete
- `vars` A table where the keys correspond to the keys in obj which should be tweened, and their values correspond to the destination.

**Listing 10** flux.lua example

```
-- Moves the ball object to the position 200, 300 over 4
 ↪  seconds
flux.to(ball, 4, { x: 200, y: 300 })
```

The rate at which a property is changed can be altered by using a different *easing function*. The most common easing function is `linear`, which simply adds a constant amount per frame. However there are 28 different easing functions in *flux* ranging from `elastic-in-out` to `expo-in-out`

[14] rxi. (2017). Flux, [Online]. Available: https://github.com/rxi/flux (visited on 12/06/2017).

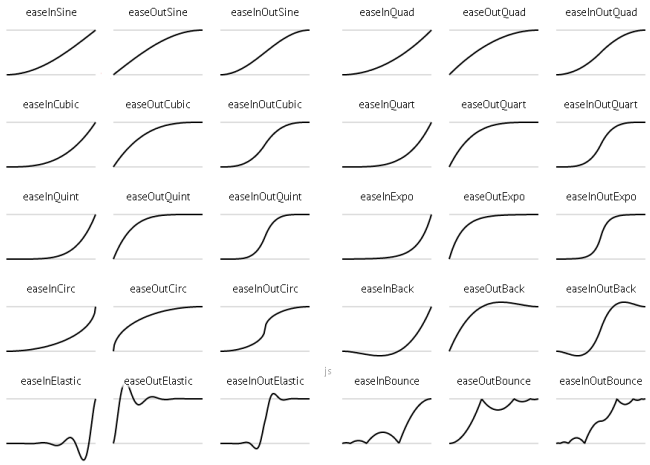



**Figure 26** All the different kinds of easing functions

Figure 26 shows all the different easing functions characteristics, with time being on the y-axis, and the value being modified on the x-axis.

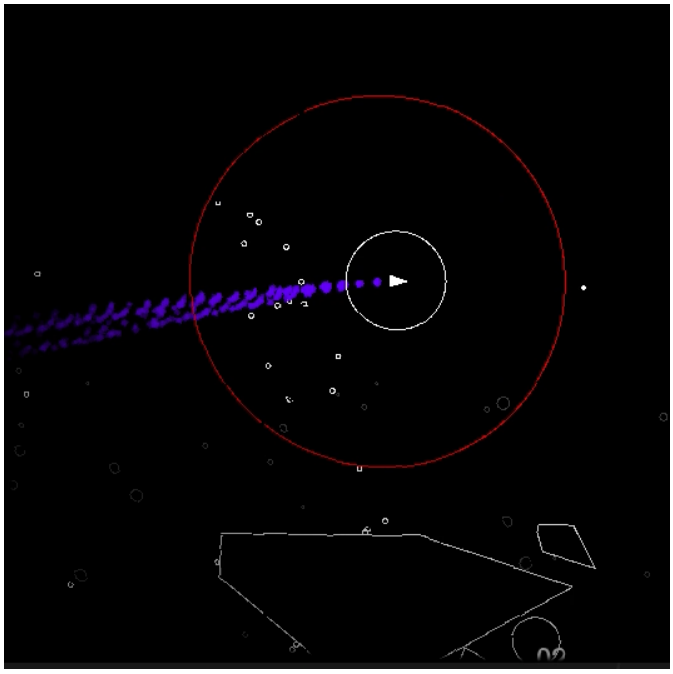


**Figure 27** Particle trails on a player ship

*flux* was used on the particles in 27 to tween the particles velocity and opacity to zero, to give the impression of the particles fading out.

### *D. Fuel, Health, Ammo, Slo-mo and Oxygen*

Adding resources to **Issx** added another layer of difficulty to the game. The player must manage all resources to maintain optimal ship performance and thus have a better chance at survival.

- **Fuel** The player has a limited amount of fuel which limits how far they may travel, moving faster consumes more fuel, when all fuel is depleted the player will move incredibly slowly, being less agile they are more vulnerable to attack
- **Health** When the player is hit by a enemy projectile, health will be decreased, when health reaches zero the player dies and re-spawns.
- **Ammo** When firing a weapon, the player loses a set amount of ammo, when ammo reaches zero, the player can no long fire.
- **Slo-mo** Slows down the game to give the player increased reaction times when in tricky situations. When exhausted it takes 5 seconds to regenerate and so should be used scarcely.
- **Oxygen** Oxygen is used up when performing actions, shooting / using boost / interacting with objects, when oxygen reaches zero the screen will start to distort and the player will lose a constant amount of health over time until oxygen levels are restored.

### *E. Pickups*

Introducing such factors on the player would be unfair if there was not a system that could increase / restore these values. Thus it was necessary to introduce pickups that the player could collide.

Pickups in the world act as normal collision objects, they cannot however, be picked up by an enemy ship. When the player interacts with a pickup, the pickup its self is set to become a sensor (thus no collision response), and the player receives a random amount of a resource.

Figure 28 shows a first iteration of an ammo pickup.

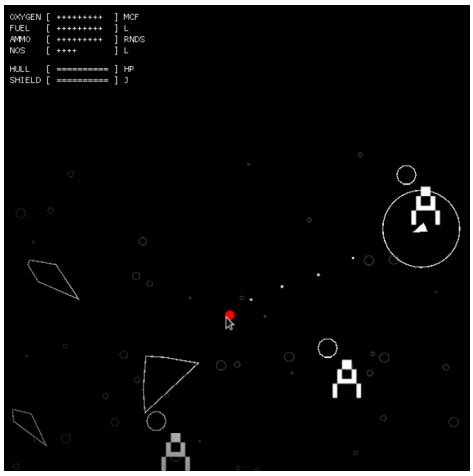



**Figure 28** Ammo pickups at initial development

Later in development pickups were changed again such that their colour related to the value of the resource that it restored. This makes it easier for the player to recognize pickups in the thick of battle.

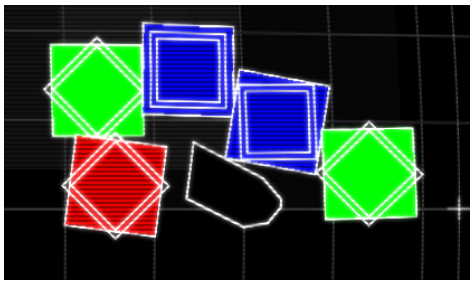


**Figure 29** Polished pickups

- **Red**, restores Ammo

- **Yellow**, restores Fuel
- **Blue**, restores Oxygen
- **Green**, restores HP

### *F. Players and Ships*

The Player is the most important aspect of any game and must be enjoyable for the player to perform actions with it. A large portion of time was dedicated to tweaking player movement, speed, fire rate other misc. parameters.

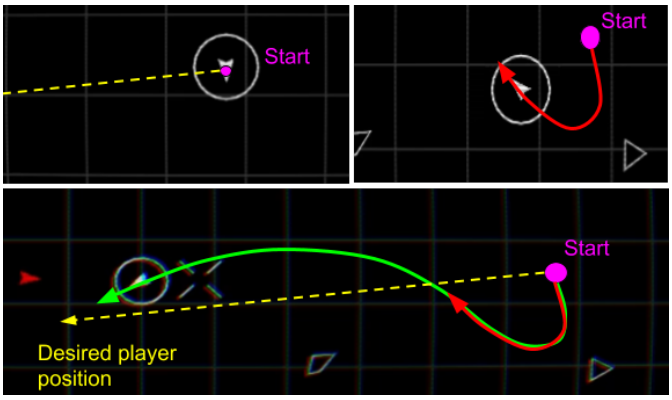



**Figure 30** Player movement explanation

The player's ship is moved by applying forces to the rear end of the ship. The angle of the ship controls the direction vector of motion. The algorithm that moves the ship applies the forces to the ship in a natural, space-like manner.

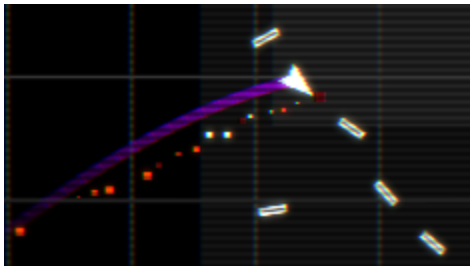


**Figure 31** The player

### F.1 Player and Game controls

- **F**: Shoot
- **W**: Boost
- **Left MB**: Slow motion
- **Right MB**: Stop movement
- **P**: Pause game
- **Escape**: Quit game

### *G. Creating a Box2D wrapper*

Mid-way through programming I realized my current way of dealing with collision, contacts and object identification was not going to fit the scale of the project, it was slow, un-scalable and required a lot of repeated code throughout classes.

To solve this issue would require a complete overhaul of the current Object Orientation tree, I took a detour from developing the game and into developing a Box2D wrapper which would solve all my problems. In my current code I was simply looking through a table for a match with the collision fixture from the `beginContact` function. This solution, though working, is terribly inefficient, running in O(n) time. My aims were simple, identify the other object within O(1) time, this can solved with the use of hash tables.

The result of this detour was **zephyr**.[15]

### G.1 Hash tables

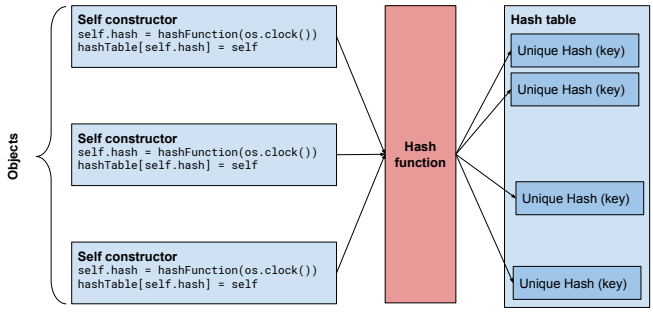



**Figure 32** A simple hash table example

A hash table is like a normal table, but instead of the object keys being numerically ordered, the table keys have hashes.

When an object is created, it is created at a different time than all the other objects since computers operate synchronously, if only a microsecond time difference between them. This guarantees a different key such that no values will be over-written when inserted into the table.

This number is now run through a hash function, the hash function transforms this piece of data into something that can be managed later on, in this case the hash function transforms our number into a *UUID*, Universally unique identifier.

A UUID is a 128-bit number used to identify information in computer systems. While the probability that a UUID will be duplicated is not zero, it is close enough to zero to be negligible.

$$\sqrt{2 \times 2^{122} \times ln\frac{1}{1-p}} \tag{1}$$

Thus, for there to be a one in a billion chance of duplication, 103 trillion version 4 UUIDs must be generated.

**Listing 11** UUID generation function

```
export UUID = () ->
  fn = (x) ->
    r = math.random(16) - 1
    r = (x == "x") and (r + 1) or (r % 4) + 9
    return ("0123456789abcdef")\sub(r, r)
  return
   ↪  (("xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx")\gsub("[xy]",
   ↪  fn))
```

This value is then set as a key in the `lssx.objects` table, acting as a pointer to the object.

Compared with the previously old code which relied on checking which fixture was ourself, and then looping through all objects to find the other fixtures key.

[15] twentytwoo. (2017). Zephyr, [Online]. Available: https://github.com/twentytwoo/zephyr (visited on 12/05/2017).

**Listing 12** Elegant beginContact function

```
Physics.beginContact = (a, b, coll) ->
  print("contact")
  -- pass a->b and b->a
  lssx.objects[a\getUserData().hash]\beginContact(b)
  lssx.objects[b\getUserData().hash]\beginContact(a)
```

The hash table method has the advantage in that we already know what objects `beginContact` callback we need to call, and we already know the other fixture that were colliding with, b, thus we can just as easily get the other fixtures table key since the fixture has a reference to it in its own `UserData`.

The base class for all objects in the game is the `Object` class, this class defines defines a creation time, creates a custom unique hash and provides a simple function to remove the object from the table.

All objects that have some kind of interaction with physics are children of `Object`, given the class `PhysicsObject`, in this class objects are given a body, and a function that updates their position, as well as another function that not only does the same as the parent class `Object`, inserts the command in the aforementioned Physics buffer to remove self from the `lssx.objects` table and Box2D world.

**Listing 13** Base object class

```
class Object
  new: () =>
    @creationTime = love.timer.getTime() - lssx.INIT_TIME
    @hash = tostring((@creationTime))\gsub('%.', '')
    -- Insert ourself into the lssx.objects table at the
     ↪  hash key
    lssx.objects[@hash] = self

  remove: () =>
    lssx.objects[@hash] = nil

class PhysicsObject extends Object
  new: (world, @x, @y, bodyType) =>
    super()
    @body = love.physics.newBody(world, @x, @y, bodyType)
    -- Leave a reference to the table key (for collision
     ↪  data)
    @body\setUserData({hash: @hash})

  remove: () =>
    -- Remove self from global table, Box2D destroy self
    Physics.addToBuffer ->
      super\remove()
      @body\destroy()
```

Two objects `Test` and `Test2` will be created for the purpose of explanation, their behaviour will be shown when they collide. They both have the same collision callback,

**Listing 14** Object beginContact example

```
beginContact: (other) =>
  other_object =
   ↪  lssx.objects[other\getBody()\getUserData().hash]
  print "I collided with object " ..
   ↪  other_object.__class.__name .. " with key " ..
   ↪  other\getBody()\getUserData().hash
```

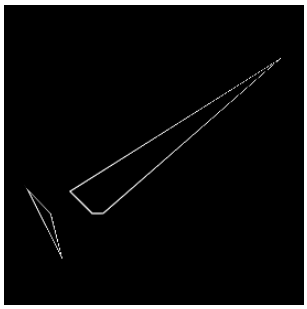


**Figure 33** Before collision

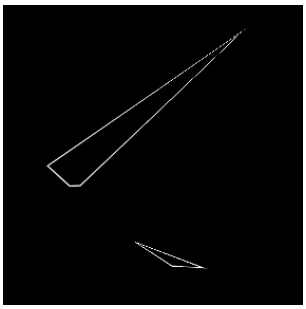


**Figure 34** After collision

When the smaller object collides with the larger object, the global `beginContact` function is called, this in turn calls both of the objects `beginContact` function which prints out what they collided with and it's key in the table.

**Listing 15** Polygon contacts

```
love.load = () ->
  Physics.load()
  -- Create two new PolygonPhysicsShapes (a custom class),
   ↪  and insert them into our Box2D world and
   ↪  lssx.objects object table
  Test = PolygonPhysicsShape({10, 20, 30, 40, 40, 80}, 0.1,
   ↪  lssx.world, 210, 300, "dynamic")
  Test.body\applyForce(10000,0)
  Test2 = PolygonPhysicsShape({10, 20, 30, 40, 40, 40, 200,
   ↪  -100}, 0.1, lssx.world, 300, 300, "dynamic")

love.update = (dt) ->
  Physics.update(dt)
  for k, object in pairs(lssx.objects) do
    object\update(dt)

love.draw = () ->
  for k, object in pairs(lssx.objects) do
    object\draw()

export beginContact = (a, b, coll) ->
  --Simply retrieve our hash, call the object that matches
   ↪  the hash's beginContact function and pass the other
   ↪  contacts fixture
  lssx.objects[a\getBody()\getUserData().hash]
   ↪  \beginContact(b)
  lssx.objects[b\getBody()\getUserData().hash]
   ↪  \beginContact(a)
```

## G.2 Custom Hashes

Sometimes we may not always want to use a custom hash, for instance we may want to set a hash of *"Player"* such that it can be easily accessed via `lssx.objects["Player"]`, in this case an arguement must be added to the *Object* class, shown:

**Listing 16** Custom hash code

```
class Object
  new: (customHash) =>
    @creationTime = love.timer.getTime() - lssx.INIT_TIME
    if customHash != nil then
      @hash = tostring(customHash)
    else
      @hash = tostring((@creationTime))\gsub('%.', '')
```

And then in the *PhysicsObject* child constructor, we call `super(...)` which initializes the custom the hash first, and then the *PhysicsObject* hash is set to the bodies `UserData`.

A typical collision between two objects, in this case an Asteroid and Player prints the following to `stdout`.

```
4.730s [important ] beginContact() triggered
4.756s [collision ] -> Asteroid, k:
↪  3c1aa946-901f-48b2-a1dd-3fd3fb984821
4.761s [collision ] -> Player, k: Player
```

The log shows the asteroid object's hash is `3c1aa946-901f-48b2-a1dd-3fd3fb984821`, a UUID and the player's hash is `Player`, a custom hash. This means that all the objects attributes and methods can easily be accessed via `lssx.objects[3c1aa946-901f-48b2-a1dd-3fd3fb984821 / Player]` since the hash is a pointer to the object's key in the global object table.

This new method of object identification will allow for scalable development of the project.

### *H. Cameras*

#### H.1 hump.camera

Helper Utilities for a Multitude of Problems is a set of lightweight yet mighty useful tools to build your game. It will help to get you over the hump.

A component of HUMP is the camera module. A camera can "look" at a position. It can zoom in and out and it can rotate its' view. In the background, this is done by actually moving, scaling and rotating everything in the game world.[16]

This is achieved by creating a HUMP camera instance and adding everything one would to be moved by the camera inside its draw function. For example:

```
function love.draw()
    camera:draw(draw_world()
    draw_hud()
end
```

The contents of `draw_world` is modified by the camera and can be translated/scaled/rotated to the programmers will, however, the HUD element, `draw_HUD`, is not modified and will stay in the same position in the window. This is done exactly the same as in **lssx**, albeit more complex.

As mentioned in the research section, camera's can simulate parallax between objects giving the illusion of depth. In order to do this more than one camera instance is required, one for each "layer". HUD parallax is described in **??**.

#### H.2 STALKER-X

STALKER-X is another camera library for LÖVE, created by SSYGEN (developer of BYTEPATH). It provides basic functionalities that a camera should have and is inspired by hump.camera and FlxCamera.[17]

STALKER-X is an improvement over the HUMP camera in that it is programmed to have "deadzones". Deadzones define different areas in which the camera will or will not follow a target. STALKER-X also has the ability to set camera lead on a target (how far it will follow ahead) and *lerp* (Linear-intERPolation) values which modified how "sticky" a camera is relative to a target.

[16] vrld. (2017). Hump camera, [Online]. Available: http://hump.readthedocs.io/en/latest/camera.html (visited on 12/05/2017).

[17] STALKER-X. (2017). Ssygen (adn), [Online]. Available: https://github.com/SSYGEN/STALKER-X (visited on 12/05/2017).

STALKER-X also has several helper functions, from `shake` to `flash` and `fade`.

- `shake`: Shakes the camera by some intensity and frequency over time `t`.
- `flash`: Quickly flash the camera contents black, useful for effects when the player gets attacked
- `fade`: Change the camera contents opacity to 0 over time `t`, useful for scene transitions.

Since STALKER-X was inspired by HUMP camera, their useage is very similar, the addition of the aforementioned functions is the reason why STALKER-X was chosen to be the main camera system in **lssx**.

### *I. Artificial Intelligence*

To keep an arcade-like style of game-play the AI was intentionally simplistic. AI is the most CPU-heavy aspect of any game, keeping low levels of intelligence allows the game to support many more entities per world, allowing for higher frame-rates and overall player experience.

#### I.1 Finite State Machine

- **Chase** If the player is within a set sphere, follow the player
- **Attack** If the player is within our FOV, fire our weaponry
- **Hide** Attempt to save ourselves by running away
- **Idle** If player is outside our max FOV, do nothing

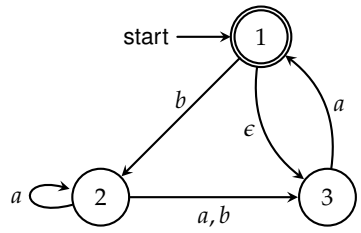



**Figure 35** Finite State Machine example

In Figure 35, 1, 2 and 3 are *behaviours*, e.g. run, hide, jump. $a$, $b$, and $\epsilon$ are *transitions*, a change in environment / conditions causes a transition to occur, thus changing the entity behaviour.

Conditions can be continually checked via a typical `case switch`, this is called *polling*.

#### I.2 Path finding

Path finding should be rather simple in nature to allow for high on-screen enemy counts (since basic path finding reduces CPU-time per entity). Considering the fact that each enemy will exist for a short duration in time, due to player actions against them, sophisticated behaviors are not required. The AI should be hyper-aggressive, placing pressure on the player to react faster to incoming waves of entities. The aim is to create an AI which is fun to play against, not technically complex.

The path finding for the AI is similar to the way in which the Player ship follows the mouse location, instead of the mouse location, the AI follow the Player location. A set of variables are introduced into the AI movement to make them seem more intelligent.

### J. Director and Game loop

The *director* controls the game loop by spawning fresh waves of enemies, increasing scores, difficulties and also manages the game over / game restart screens.

### K. UI

UI is specifically simple to keep player distraction down to an absolute minimum. The UI should only display important information about the player's progress and current state.

The UI consists of several elements:

- **HUD**, displays player resources
- **LineExplosion**, Procedurally generated explosion effect
- **FlashSq**, another PG explosion effect, less subtle
- **Cross**, Miscellaneous UI for HUD and background

#### K.1 LineExplosion

A LineExplosion is a minimal explosion effect which are similar to the one's one in **DATA WING**, the LineExplosion is useful for showing how procedural generation actually works.

```
class LineExplosion extends Object
  new: (@x, @y, @count=math.random(4,8), ...) =>
    super(...)
    @c = { -- ox, offsetx, ex, endx
      ox: 2, oy: 2
      ex: 10, ey: 10
      o: 255
    }
    t = math.random(5)/10+0.1
    mo, me = math.random(20)+5, math.random(20)+10
    flux.to(@c, t, {ox: mo, oy: mo, ex: me, ey:
     ↪  me})\ease("quadout")\oncomplete(-> super\remove())
    Timer.after(t/2, -> flux.to(@c, t/2, {o: 0}))

  draw: () =>
    love.graphics.setColor(255,255,255, @c.o)
    for i=1, @count do
      PushRotate(@x, @y, math.rad((360/@count)*i))
      love.graphics.line(@x+@c.ox, @y+@c.ox, @x+@c.ex,
       ↪  @y+@c.ey)
      love.graphics.pop()
```

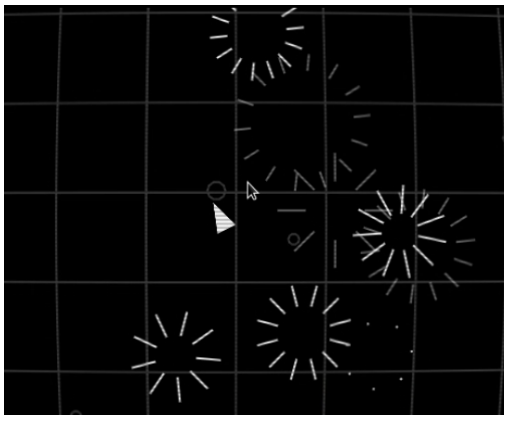


**Figure 36** Different types of LineExplosion's

#### K.2 HUD

HUD or *Heads Up Display* shows the player information, specifically related to their resources, score and number of kills.

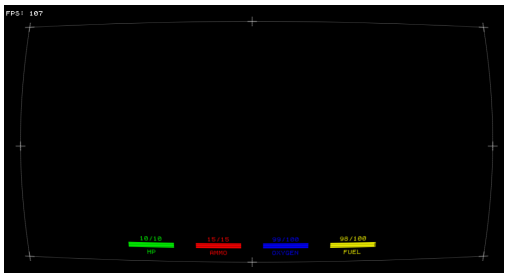



**Figure 37** Player HUD

The HUD is constructed out of four HUD bar's, given the class `HUD.elements.bar`.

The HUD also has a subtle overlay, similar to that present in **DATA WING**, the overlay is present only for aesthetic purposes and gives the impression of the player looking at a CRT computer display (which is helped greatly via the use of **Moonshine** CRT & scanlines shaders). The overlay is constructed from a series of lines and `Cross` objects, which were specifically created for the HUD.

The HUD overlay features parallax, the effect whereby the position or direction of an object appears to differ when viewed from different positions, e.g. through the viewfinder and the lens of a camera. The parallax works by creating a new **STALKER-X** camera instance and creating an offset from the screen center by adding the player's linear velocity (in the x, y position, which includes negative numbers)

The camera then uses linear-interpolation to move to the new X, Y position over time (time taken is calculated by a discrete *lerp* value).

```
HUD.update = (dt) ->
  x, y = HUD.player.ship.body\getLinearVelocity()
  HUD.camera\follow(550+x/15, 300+y/15)
  HUD.camera\update(dt)
```

This has the effect of the entire HUD shifting slightly away from the player's forward vector, this gives the impression of depth as the movement between player and HUD is relativistic.

### L. Projectiles

A projectile is composed of a body, fixture and shape. The very nature of projectiles means they move at a typically faster velocity than most other objects in the game, this can introduce certain issues when using a fixed time step physics engine,

There are two methods to check for body collisions:

- at their location when the world is updated (default)
- using continuous collision detection (CCD):

The default method is efficient, but a body moving very quickly may sometimes jump over another body without producing a collision. A body that is set as a bullet will use CCD. This is less efficient, but is guaranteed not to jump when moving quickly.[18]

In Box2D, collisions are detected every tick at the point where the object moves to. If the object moves into another object, then the physics engine pushes it back to where the collision happened initially, as shown in Figure 38.

The process of moving an object back to will never actually seen as that is corrected during each tick, before the draw function is called. However, if our projectile was moving so quickly that it simply passed through the grey square, no collision would be detected, and

[18] Rude. (2017). Body:setbullet.

the physics engine would not be able to correct the balls position. This is known as tunneling.[19]

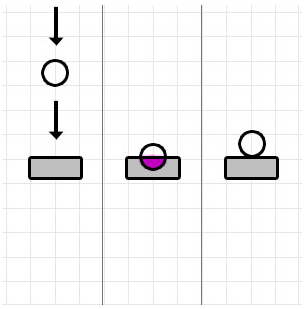


**Figure 38** Without CCD

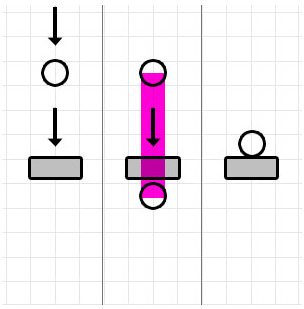


**Figure 39** With CCD

In order to apply CCD to objects,[20]

```
@body\setBullet(true)
```

Continuous Collision Detection (CCD) prevents cases of small, fast-moving actors from traveling through thin walls. CCD impacts performance. As such it should only be used on objects that need it.

### *M. Components*

#### M.1 Emitters

Emitters are simply projectile creators, they pass horizontal and vertical components to the projectile.

An interesting bug that arose during development was that bullets seemed to slow down then a ship was traveling at high speeds, however, this was just effect of relativistic effects. On creation, a set amount of force was applied to the projectile, the projectiles were traveling at a constant velocity, but compared to the ship they appeared slower. This however does not coincide with the laws of conservation of momentum. A projectile should in-fact inherit the velocity of the ship.

$$\rho = mv \tag{2}$$

$$\rho_{proj} = m(v + v_{ship}) \tag{3}$$

$$\rho_{proj_x} = m(vcos(\theta) + v_{ship}cos(\theta)) \tag{4}$$

$$\rho_{proj_y} = m(vsin(\theta) + v_{ship}sin(\theta)) \tag{5}$$

Emitter can emit any number of projectiles during an emission, they can also control the spread of projectiles by passing a different dy/dx, as opposed to straight ahead, this allows the emitter component to simulate any kind of firearm.

In the effort to reduce complication, the emitter does not handle a list of its emitted objects, it instead inserts objects into the global object list, `lssx.objects`.

#### M.2 Shields

Shields are fixtures that attach to the Ship body in components, a shield has a GroupIndex of either Friendly or Foe, a shield will only have a collision response with an object of category Projectile.

Projectiles will have a different GroupIndex depending on if the player or AI has fired it from their emitters, this prevents the shooter from damaging their own shields as shields project outward from their body.

A shield has a set HP, when this HP is reduced to zero, the shield deactivates for a set amount of time depending on the size of its health. i.e. a Shield will take longer to be restored if it has a large HP than one with a small HP.

### *N. States*

A gamestate encapsulates independent data an behaviour in a single table. Different states can be used to separate different sections of the game from each other, such that their data is private and does not leak.

A typical game has the following states:

- Main Menu
- Game
- Pause
- Game Over

In *lssx*, this is simplified such that their are only three states:

- Splash
- Main Menu
- Game

#### N.1 Splash

The splash state is entered as soon as the game is entered. The splash state uses the Splashy[21] library, created by videah.

> **splashy** is a simple and basic library for LÖVE, that allows the easy implementation of splash screens to any project.

The Splash screen also sets the `lssx.FIRST_TIME` flag true, which is used later in the game loop to decide if the instructions should be shown again.

#### N.2 MainMenu

The MainMenu state is simply a scrolling text screen that gives time for the game to load it's assets.

#### N.3 Game

The game state is more of a "God" object, as in it combines a lot of functions and states (pause, lost-focus, game-over) together - which is generally considered bad practice but can be passed due to the small scale of the game.

The game state contains a **Director** that manages the objects and entities in game. The Director also manages the player score and game over screen.

### *O. Documentation*

#### O.1 lssx

lssx "God" table, contains many constants that are used throughout the game.

```
require("lssx")
```

- **lssx**, `table`
  - **objects**, Global object instance container, `table`
  - **categories**, Box2D categories, `table`

[19] Stencyl. (2017). Continuous collision detection (ccd), [Online]. Available: http://www.stencyl.com/help/view/continuous-collision-detection/ (visited on 12/07/2017).

[20] Rude, *Body:setBullet*.

[21] videah. (2018). Splashy, [Online]. Available: https://github.com/videah/splashy (visited on 03/07/2018).

  - **groupIndices**, Box2D group indices, `table`
  - **INIT_TIME**, Game initialization time, `number`
  - **CAMERA_ZOOM**, STALKER-X game camera zoom, `number`
  - **WIDTH**, Scaled window width, `number`
  - **HEIGHT**, Scaled window height, `number`
  - **SCALE**, Global game scale, `number`
  - **W_HEIGHT**, Actual window height, `number`
  - **W_WIDTH**, Actual window width, `number`
  - **PLAYER_DEAD**, Player dead indicator, `boolean`
  - **FIRST_TIME**, first time opened indicator, `boolean`
  - **PAUSE**, Game paused indicator, `boolean`
  - **TITLEF**, Title font object, `font`
  - **TEXTF**, Text font object, `font`
  - **SCORE**, Player score, `number`
  - **KILLS**, Total player kills, `number`
  - **SPFX** Special Effects, `table`
    * **CHROMASEP**, Chromatic aberration strength, `number`
    * **CHROMASEP_ANGLE**, CA angle, `number`
  - **masks**, Box2D masks, `table`

### O.2 Object

```
Object(customHash)
```

- **customHash**: Define a custom hash for the object in `lssx.objects`, `string`

### O.3 PhysicsObject

Extends **Object**

```
PhysicsObject(world, x, y, bodyType, customHash)
```

- **world**: Box2D world
- **x**: x position of physics object, `number`
- **y**: y position of physics object, `number`
- **bodyType**: the type of body, `string`
  - `"static"`: Static bodies do not move.
  - `"dynamic"`: Dynamic bodies collide with all bodies.
  - `"kinematic"`: Kinematic bodies only collide with dynamic bodies.

### O.4 PolygonPhysicsShape

Extends **PhysicsShape**

```
PolygonPhysicsShape(points, density, world, x, y,
↪  bodyType, customHash)}
```

- **points**: Table listing vertices's (`numbers`), `table`
- **density**: Density of fixture, `number`

### O.5 CirclePhysicsShape

Extends **PhysicsShape**

```
PolygonPhysicsShape(radius, density, world, x, y,
↪  bodyType, customHash)
```

- **radius**: radius of circle, `number`

### O.6 ChainPhysicsShape

Extends **PhysicsShape**

```
ChainPhysicsShape(points, density, world, x, y, bodyType,
↪  customHash)}
```

- **points**: Table listing vertices's (`numbers`), `table`

### O.7 Ship

Extends **PolygonPhysicsObject**

```
Ship(world, x, y, bodyType, customHash)
```

Mostly generated procedurally.

### O.8 Asteroid

Extends **PolygonPhysicsObject**

```
Asteroid(x, y, customHash)
```

- **x**: x position of Asteroid, `number`
- **y**: y position of Asteroid, `number`

### O.9 Entity

Extends **Object**

```
Entity(hp, customHash)
```

- **HP**: Health points, `number`

### O.10 Player

Extends **Entity**

```
Player(ship, hp, customHash)
```

- **ship**: Instance of class `Ship`, `object`

### O.11 Enemy

Extends **Entity**

```
Player(ship, hp, customHash)
```

- **ship**: Instance of class `Ship`, `object`

### O.12 Projectile

Extends **PolygonPhysicsShape**

```
Projectile(x, y, points, groupIndex, customHash)
```

- **groupIndex**: Box2D body group index (collision masking), `number`

### O.13 Bullet

Extends **Projectile**

```
Bullet(x, y, damage, customHash)
```

- **damage**: Damage to deal to entity upon collision, `number`

### O.14 Pickup

Extends **PolygonPhysicsShape**

```
Pickup(x, y, customHash)
```

### O.15 Shield

Extends **CirclePhysicsShape**

```
Shield(hp, x, y, groupIndex, customHash)
```

### O.16 Cross

```
Cross(x, y)
```

### O.17 FlashSq

Extends **Object**

```
FlashSq(x, y, intensity, customHash)
```

- **intensity**: Size / brightness of Flash square, `number`

### O.18 LineExplosion

Extends **Object**

```
LineExplosion(x, y, count)
```

- **count**: Number of lines extruding from center, `number`

### O.19 HUD.elements.bar

```
HUD.elements.bar(x, y, pointer, value, color, bgcolor)
```

- **pointer**: Reference to object being monitor, `object`
- **value**: Value of object to be displayed (e.g. hp, ammo), `string`
- **color**: Table containing rgb values for foreground bar colour
  - **[1]**: Red, `number` (0-255)
  - **[2]**: Green, `number` (0-255)
  - **[3]**: Blue, `number` (0-255)
- **bgcolor**: Table containing rgb value for background bar colour

## 4. Play-testing and improvements

### A. Q&A

Eight people play-tested *lssx* in its current state.
The following questions were asked:

- What do you like / dislike about lssx?
- What would you like to see added / removed or changed?
- Rate the game in its current state.

**What do you like / dislike about lssx?**

- I really enjoy the high learning curve, as well as the necessitation of gathering different resources so that the player cannot simply focus on one singular task. I couldn't entirely tell, but it seemed that there was only one 'tier' of enemy, so if that were the case, I would dislike that aspect of the game.
- you're killing soviets and also the aesthetics. as for dislikes, the rather jarring shooting sound.
- Shooting sfx is awful and is played way too much. The camera tween seems like it would make me want to puke after a while. I do like the shaders and the hex theme.
- It looks a lot like asteroid.
- It goes at a nice pace
- it looks like bytepath
- Screen shakes too much
- landed the aesthetic well, sound balance doesn't seem right, hard to make-out feedback/kills noise from gun noise

**What would you like to see added / removed or changed?**

- I would like to see different 'tiers' of enemies added, along with better rewards for more difficult ones. Maybe even a 'boss' enemy after enough kills. It would also be interesting to see the usually useless asteroids occasionally contain something useful within them - perhaps some gun upgrade from a wrecked ship. Finally - and this is certainly the game's most pressing issue
- the shooting sound.
- maybe make it a bit slower?
- Smooth out the zooming. Maybe more colors or even some blooming on certain colors?
- multiplayer
- Halve the screenshake

### B. Q&A results

Players rated the game in its current state a 8.5/10.

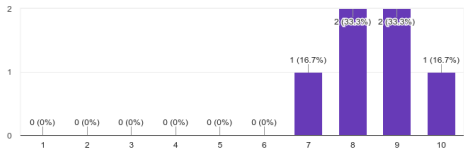



**Figure 40** lssx itch.io change-log

### C. Improvements

- Changed shooting sound
- Reduced camera shake
- Increased difficulty via modifications to Director spawning code
- Created separate removal buffer for Box2D destroy calls
- Added several new SFX
- Added pause screen
- Fixed memory leaks from new states
- Fixed crashes from zephyr attempting to access nil objects
- Fixed bullet clipping shield
- Added bloom shader
- Added kill streak indicator
- Added time indicator to HUD
- Fixed timers being affected by slow motion
- Added LÖVE splash
- Modified Asteroid destruction algorithm
- Modified low HP enemy AI behaviour to spin out of control
- Changed background music

## 5. Publishing

### A. itch.io

itch.io is a website for users to host, sell and download indie video games. Released in March 2013 by Leaf Corcoran, the service hosts nearly 100,000 games and items as of February 2018.[22]

With the modifications added to **lssx** it was finally ready for the game to be released. itch.io provides a simple website building service that allows users to advertise their games.


**Figure 41** lssx itch.io page

[22] Leafo. (2018). Itch.io, [Online]. Available: https://en.wikipedia.org/wiki/Itch.io (visited on 03/27/2018).

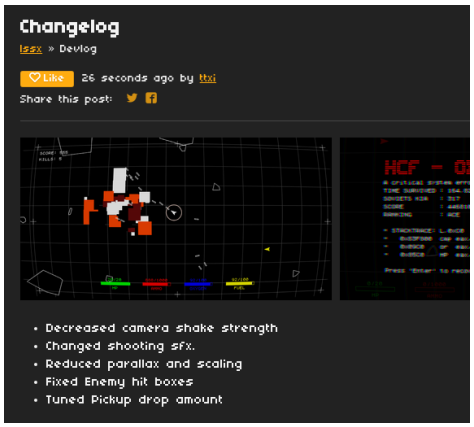



**Figure 42** lssx itch.io change-log

**lssx** has been packaged for Windows (32 bit) and Linux.
**lssx** can be found at: https://ttxi.itch.io/lssx

## 6. Project overview

### *A. Final thoughts*

Overall pleased with outcome, achieved my goal of making a short enjoyable game with a fair amount of re-playability. Gained a strong understanding of *Box2D* and appreciation for simple, elegant solutions to problems.

Things learned:

- Physics is hard
- Desire to work on a project / code quality drops off exponentially over time
- 90% effort required for the last 10% of work

After EPQ I'd like to further develop **zephyr** make it easier for people to develop their own games.

Links related to **lssx**

- **lssx beta**: https://github.com/twentytwoo/zephyr
- **lssx**: https://github.com/twentytwoo/lssx
- **blog post about lssx**: https://ttxi.gq/posts/lssx-and-lessons-learned
- **lssx LÖVE thread**: https://love2d.org/forums/viewtopic.php?f=14&t=85003
- **lssx presentation**: https://ftp.ttxi.gq/5cTO0kDM0Y.pdf

## 7. Screenshots

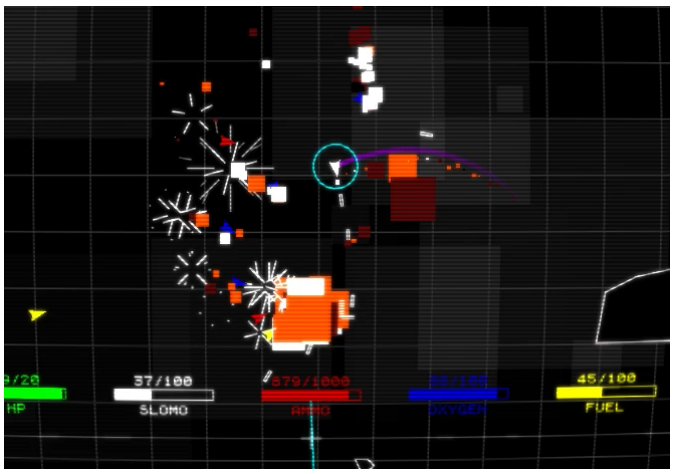



**Figure 43** Screenshot 1


**Figure 44** Screenshot 2

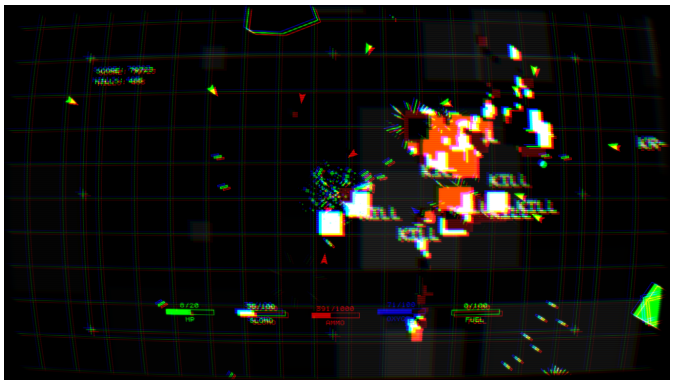


**Figure 45** Screenshot 3

## 8. Commit log

| Date | Commit message | URL |
|---|---|---|
| Nov 4 18:11:42 2017 | Create README.md | 8f214a13b128848678d7b1e6f20144ceea7f741b |
| Nov 5 15:36:31 2017 | Box2D is overkill | d92d0bacb3852023bd00334a2698054cd10bed4b |
| Nov 5 19:32:30 2017 | Basic controls | 1857b9ddd0bbfdb101bb2d9251ea5ed1def53178 |
| Nov 5 21:35:25 2017 | Basic guns projectiles | e7fd04db3cdbac51dbfcf41f0522be91fe1f8db0 |
| Nov 6 12:26:37 2017 | Bullet math and camera complete | 652f32e268e735c06fbf7aefaf2e8bd1637b3d6c |
| Nov 8 19:59:22 2017 | Fleshing out understanding of love.physics removed windfield because it s too limiting one body fixture and shape will place too much strain on CPU later also does not allow multiple fixtures per body | a00f7b658f972aeaa7c30127242b899dedcccdb8 |
| Nov 9 14:57:47 2017 | Rewrote command module physics structs | bdcef82a995b57aee20ff9565f18dfcda9b3d72d |
| Nov 9 14:59:32 2017 | Removed STI | dca3ce860580ef2d373b3e29480411b16bddae4a |
| Nov 9 23:32:15 2017 | Using a buffer for world changing via box2d callbacks | 17db76d214b689ac3b37ee38886b79ed22fe2de5 |
| Nov 9 23:33:44 2017 | Fixed merge | 9fca53a4030621fc9bda086ed749a66a10a13fab |
| Nov 10 10:15:16 2017 | Update README.md | 69d8cb1233b7832eb91fecdd902c65c49a4f7b07 |
| Nov 10 11:36:48 2017 | Progression in shape clicking | ef3ae5f3312ec913382b88df94bbdb19129c3c20 |
| Nov 10 11:38:36 2017 | Progression in shape clicking | dec4c6b92be6243f4f1c49e7100a0b8a6ee2d9d7 |
| Nov 10 11:38:53 2017 | Merge branch master of https github.com twentytwoo lssx | 06028e2d14565a791f817083075f8e1513630314 |
| Nov 10 11:40:41 2017 | Progression in shape clicking | 733de1c664feefc4236306350fecbdf23e9a60f4 |
| Nov 10 11:41:35 2017 | Progression in shape clicking | 618b897b4e1287c0a86b1ccbe8bb2d8ca8bbb9c9 |
| Nov 10 13:15:32 2017 | Started on recursive shape connection algorithm | 42ce5e7622a38fdbac1404892ede65d5cba04e76 |
| Nov 10 13:16:19 2017 | Started on recursive shape connection algorithm | 201e0a429a07dbd4d706f39b8e554e5898f9d21d |
| Nov 10 23:24:21 2017 | Problems with shape connections | 3fcedc6cc6351a12f272c5cfff1f42f8852be827 |
| Nov 11 18:22:22 2017 | Working on adding rotation to vertices | 15b2dd27d467a83bde191b64c90b80f909d536d6 |
| Nov 11 20:34:23 2017 | Fixture adding algorithm works kind of needs touching up to make modular | e42319851d38eba8bd7198e19a06ecde79ec730a |
| Nov 11 22:21:43 2017 | Cleaned up shape connect algo | d9cd9672d5124f12ed350bbba21b54aa456b9d9a |
| Nov 12 13:01:10 2017 | Connection algo works doesn t detect for invalid connections other shape overlaps toCM and resolves it to rotate them | b7cf5e7f9e56fd3f24eedfaee66d4cb92fbdee43 |
| Nov 12 14:08:37 2017 | Scrapping one body multiple fixtures because of difficulty to implements bugs the fact that turret like weapons would be impossible to create later on because fixtures inside one body cannot move independently to on another | bebacbafeecd68c9ee87693bf2b94cae7f2a85f7 |
| Nov 12 15:12:37 2017 | SINGLE BODY MUTLI FIXTURE PHYSICS | 17465f46a8c232ef20153e5bf9900d56f8842f56 |
| Nov 13 00:10:46 2017 | Ship control code mostly done | 8282f59fde0bd43e7cfebf109d846d59fd5c9a59 |
| Nov 14 12:30:49 2017 | Camera work done | b32f08f37a8768f6700fa62979dcff1eaff83446 |
| Nov 14 18:36:16 2017 | Added a console interface setup LoverNet | feaf1c9f72dec21e3ab2182607b0c1db16bc49b2 |
| Nov 14 18:49:24 2017 | Added empty files for most of game files | 97d5aef4e743eac2a700167bb94ace80dd41aadd |
| Nov 15 14:43:23 2017 | Server Client set up | af7c43eb3cb3448df0ca8dbadc9d0da990cb9675 |
| Nov 16 01:02:01 2017 | Emitter bullet and projectile classes working | 2a53213e8bf3fd52d30db68a0863d840b9a12ad5 |
| Nov 16 18:01:14 2017 | Proceedural asteroids camera s emitters | ff8a678c4fe7acfa4a2023334a884747eec976b6 |
| Nov 17 00:37:27 2017 | Emitter emitting projectiles at self position correctly now | a5a9cb8e61b02a0b802cdd929cb6772189620bfd |

| Date | Commit message | URL |
|---|---|---|
| Nov 17 23:45:18 2017 | Replaced HUMP camera with STALKER X added Player class | 99411df9801d342f76146c959ca1b8788f335c26 |
| Nov 17 23:45:50 2017 | Removed old file | 1df36033493bfdafeeecdfc7b56e0ef305843eea |
| Nov 18 00:41:24 2017 | Started on Sheilds and Missiles | b75fe6a15dd5e9c853df99bb7f1ec084510d5164 |
| Nov 18 11:51:58 2017 | Fixed Debugger | 616df329dd491409cd0f8f4f6ec3a9e151dc419d |
| Nov 18 13:34:50 2017 | Modularized ship components shields emitters etc | e7fd58f9c3d96e20a737e78bff818f41263a46e6 |
| Nov 18 15:17:11 2017 | Added ammo limit for players emitters added moonshine shader effects | bb89c55c96d6f13fb4e30881cba590715e35968d |
| Nov 18 19:33:31 2017 | Started on ammo o2 fuel pickups added shield disabling tweening Player HUD | 3d17f229bd89cb62d407e75828b80967e82d78d6 |
| Nov 19 00:41:20 2017 | Oxygen Fuel Ammo NOS integrated each affects another Player class updated to control ship Ship can now be modularized for AI | 3b6e2baefbe95710468391777e95d9cface9ee9b |
| Nov 19 03:21:11 2017 | Started on primitive AI | 55b78fa3c15411f4428b4b8937ebfdcb5bc6bcbb |
| Nov 19 15:06:13 2017 | Pushing last nights changes taking the day off | e43e00391842725e87ac98687f456bb769344f84 |
| Nov 21 12:41:41 2017 | Collision callbacks sort of sorted out with player Ammo example Player can now pickup ammo and it add s to their ammo count | 4289b2ed8329bcef64231817d2adde7672c8c825 |
| Nov 22 01:57:29 2017 | Added ship direction reticle | 95fce4249422f889fc458fba887ca5dbce33d0c2 |
| Nov 24 12:40:01 2017 | Added player collision detection awful solution for bullets because of emitter self projectile table needs rethinking | 7e700751ec9a1777e36735deade089b313d0deee |
| Nov 25 17:06:19 2017 | Particles | b13c05911e9fbd531e9175e8cbdc2e3f19068332 |
| Nov 26 20:36:23 2017 | trails and some more AI work | ec6ec90069931964fecc42a1eaffa868897c473e |
| Dec 3 18:07:08 2017 | modified components AI.moon | 15cc2742d8f62b9849a7c74bdd607c7bf81d2eee |
| Nov 27 01:50:57 2017 | Create README.md | 24cc646f26d911ef747f68d245a9c5dab5e53a77 |
| Nov 28 12:40:46 2017 | Create main.moon | b1bc6a218a8e4cf889628ca5895c46c6065a4eb5 |
| Nov 30 13:10:40 2017 | Update main.moon | 7389001adabdaee8e1cf99bd5a2973aa385128a3 |
| Nov 30 13:25:17 2017 | Update main.moon | 67f218e5d6cb9894c34282d8136c717c45e53f60 |
| Nov 30 13:29:24 2017 | Update main.moon | 51b341e7546b9a5de9655a5b6eafa1cc9a06fa39 |
| Dec 3 01:52:56 2017 | Restructed collision callbacks to make use of hashtables other fixture table key now found in O 1 time | ec829bd9fca4a7c10ec1a23a31201e52b0c590f4 |
| Dec 3 14:22:42 2017 | Reorganized file structure | b03f751e7cd0f078f218c5c97470498b5024ecd4 |
| Dec 3 23:48:38 2017 | Reintegrated players entities ships circleshapes as well as libraries | a5634da8c5ab188840bf323c31e3aa2427aa5f4f |
| Dec 7 19:49:20 2017 | Readded player control background and particles to new engine | c20881e595e610930192130290e9e4607b9d606b |
| Dec 7 19:50:59 2017 | Correct ignoring | 46b23de561c0bcbcb6ccd9b01261abadb3aae98e |
| Dec 8 19:18:32 2017 | Readded emitters and bullets diamond problem with entity ship polygonShape and projectile bullets physicsShape | 66ef83ddcc9b4597d4db7e41238a1a8c1fcca74b |
| Dec 8 22:45:08 2017 | Resolving diamond issue with projectiles going to use polygons only for projectiles creating a sort of circle via many vertices | 194395c905a76a62766403841f9744aad9a9d092 |
| Dec 9 15:31:22 2017 | Added documentation | a2ea4c7b8795186f1dd5415ceeed9746838a9e56 |
| Dec 9 15:32:46 2017 | Fix docs formatting | 08da4e708c29e13e100e1073620019d5f1d52dd5 |

| Date | Commit message | URL |
|---|---|---|
| Dec 9 15:33:34 2017 | Fix docs formatting | 8f590509f729d28eef64f9c8792e7db8235e313e |
| Dec 19 12:40:00 2017 | Updated Emitters to working state started working on group indexes restructed Projectiles and Bullets added octogonal bullet shape projectile lifetimes working | c54a64f330f1587b5b53723f61c337d028bd711d |
| Dec 20 14:31:36 2017 | Group indexes between player bullet working started on readding shield to zephyr engine | 1b45389038dbe8a86bdce7d8df426a256e6058dc |
| Jan 30 12:20:39 2018 | Create EntityManager.moon | d9a09db3542581a7f1c81272abc428c9f8a81a7f |
| Jan 30 12:20:53 2018 | Update main.moon | 65e6fa55d805c0aa6a8f974662b2694ace4db4b6 |
| Jan 30 12:22:36 2018 | Update main.moon | 27c964ab781660752cbd9d69bee932013c3b4a3d |
| Jan 31 12:57:49 2018 | Update README.md | 61024ae098d457d7b962e6703218bbe5f856cfcc |
| Jan 31 12:58:19 2018 | Update README.md | 8bc6e8e3c33c13b7c6b560b81a060657f74965fc |
| Jan 31 12:58:52 2018 | Update README.md | dcf56487920c084730095290524e2bfcfd2f1562 |
| Feb 12 14:24:35 2018 | semi infinite grid SPFX readded camera tweaking started on enemies again | b26c56138b87043441692a4e4736621ed9e99385 |
| Feb 12 14:42:32 2018 | fixed merge issue added gamestates | 7e1c041733efb192489e354537f7325ec6fd7fe1 |
| Feb 12 15:20:21 2018 | AI pathfinding working! | b67f41695792c41b5879bc898e521f844ecaf18d |
| Feb 12 16:56:12 2018 | Added splash screen | 0a435b2254de655d01f1ee600a902ccbbd39c19f |
| Feb 12 18:17:25 2018 | finite state machine general ai stuff background stuff | 084af3a5a39a48462a975156a248fd80d7790c11 |
| Feb 13 18:48:59 2018 | fleshed out splash main menu screen | 1a235a2319c3c41674c8c9ad6e7461035b3703ea |
| Feb 13 19:51:48 2018 | Fix enemy physics removal bug | 28e23f62e335fe7c7753fa1e2d6da9c0ba3b829f |
| Feb 13 22:53:51 2018 | added sound effects modified hiding ai state | 485e0b2f586abeaf70c7f3871a0983df8f901646 |
| Feb 13 23:55:11 2018 | Reworked logging | 2d81013c30f02834d5e568fa58079c557cc9f541 |
| Feb 14 00:13:44 2018 | bug identified either race condition or repeats in the buffer | e8c4accb14a3f0941878600b9fc790b5e5baf20d |
| Feb 14 01:39:43 2018 | seemed to have fixed the bug wherein multiple body destroy calls were made throwing an error if an bullet collided with two objects in the same world tick | 7ad004ec9e54d4933d34f20d59f8de9873e5845a |
| Feb 14 02:02:22 2018 | AI tweaks | 7df2d55d3c084d8dd6eb38e1cb5bdedb41fbcaff |
| Feb 14 20:04:23 2018 | Added HUD w parallax added crosses FlashSq effects updated ai behaviour | eb9d076c42155fea73acb6da2c42dad81f7393ca |
| Feb 15 00:42:48 2018 | Added Director to control game loop and manage game over screen added UI for player hp ammo oxygen other things | 36ac8e49fc7f2c96a8ab226699da62069b8269a3 |
| Feb 15 01:05:49 2018 | tweaked player movement by changing inertia | 17c81fccacfe1e0d3b2cda8194af5d8ce7ea843c |
| Feb 15 18:53:22 2018 | fixed shields added destructable asteroids algorithm got categories mask group indexes semi working | b3997979735f2a8ec6779def07b09038686b7e87 |
| Feb 15 21:08:58 2018 | gave AI ability to shoot at player when within FOV needs more tweaking. bug Shield doesnt turn off completely | 33795839c2bfa2a94a261e3dca8e8415db9648ea |
| Feb 16 02:18:28 2018 | Attempting to fix physics bug where body deleteed and begin contact occurs attempt to get body of deleted object throws nil err added ranking to Director | 216e5ca2f0e33e142a5ca278d9d5b851aef1115c |
| Feb 16 17:08:02 2018 | improved astroid breakup algo fixed physics bug mentioned previously was to do with player and ship not being removed in same work tick because not added to pBuffer | de0e8293d517b0fd1182c3efa611761bf5f0a63b |
| Feb 16 17:10:25 2018 | removed old code from object | 414443239a193085427c6de03ecae479e7b8e6e1 |
| Feb 18 21:31:01 2018 | Added stat pickups and rot scale functions | 67eb76273e077a54b561b6dafa73b5e99731ce12 |

| Date | Commit message | URL |
|---|---|---|
| Feb 19 20:26:17 2018 | Added LineExplosions | b4471c5b455c9664c68863ce2da57580c841b2bd |
| Feb 20 01:26:54 2018 | Added some SFX tweaked some classes | 7c667ef486e5ac5029cb353a7291b5abbbeeb342 |
| Feb 21 19:06:23 2018 | probably done | 40041e155389faa5f95039e62e76f1bfca1d7eeb |
| Feb 21 19:07:05 2018 | Update README.md | aece3cc58cf0b594bc0a27a5feee264d03d0ecff |
| Mar 7 13:35:28 2018 | Delete lssx.tex | 09d3b06227b5befe3f9ea190e8740c17b9690609 |
| Mar 7 13:36:12 2018 | Update README.md | 6c3fc5bd7cf0c7793f0a92aaaa99f25111d1c222 |
| Mar 7 13:36:54 2018 | Update README.md | 6267abcaf8815331153844b6e1620546ed3ef653 |
| Mar 7 17:24:04 2018 | Added more sound effects edited Bullet Pickup main Debugger and Director | 1253dc8344d37870c5802f41ee116026b79fe946 |
| Mar 7 17:24:18 2018 | Merge branch master of https github.com twentytwoo zephyr | 9c7953d914a8f8fb4bbf92bb0c6a22367bb8e08e |
| Mar 11 21:58:18 2018 | Game reset works player can respawn without restarting game globals mostly cleaned | 6d2ba3bc75a0cdd59602682001e2fd1b2e3c5bf1 |
| Mar 13 00:51:36 2018 | Lots of polish added | 2155d9990fd265c89aa07255916ee7b2bc4ba972 |
| Mar 13 01:28:34 2018 | Director code almost done, added enemy spawning over time, modified pickups | b83fedff116641ac7c8d35beb78baad2f3386975 |
| Mar 13 18:35:30 2018 | More director mods, spawns pickups and asteroids now | c668dcf2dd57af2f3d9ed855b3bb71f0941ca673 |
| Mar 13 19:14:45 2018 | Update README.md | c0baba75638bd99684162bf23a3705edb8f21f0f |
| Mar 13 22:32:09 2018 | rewrote position code for ui such that game can be played fullscreen | 73886e5694c23f9496762e5736675edc2e1c039d |
| Mar 13 22:32:22 2018 | Merge branch master of https github.com twentytwoo zephyr | b0a91c962b68e1d7867bbeb184f40651e705b1ad |
| Mar 15 21:50:59 2018 | separate removal buffer hopefully removes box2d nil obj errs | a14871a5d1a8a9f2e33059cc92d4897eafb6b003 |
| Mar 15 21:51:40 2018 | fixed syntax bug | 35e6fc2ef0ffa813190b4b60c2c4541e2110831a |
| Mar 15 22:00:11 2018 | Balanced pickups to give more or less resources depnding on remaining value | 640e48481c1dbb4a2b30661b219aaec068339697 |
| Mar 15 22:43:37 2018 | time survived fixed added correct pausing previously timers fluxers still being updated added a basic pause screen | 5132ccce77b9c7f9f87507a9c3c5b6f4311292f5 |
| Mar 16 00:08:50 2018 | Sent Asteroids to family therapy improved their parent child relationship added scoring stats corrected time in match in gameover | 5013bc051643dcdf10fe59124684b7526322e3e7 |
| Mar 17 02:38:09 2018 | added love splash | 8bfad19899b892002df2d186cad944f091157f02 |
| Mar 17 20:18:40 2018 | Update README.md | d8ab25152cab7f9ca175e84d76e86616f56d60dc |
| Mar 18 22:06:00 2018 | Added slo motion | 5ac90da1cfae7f9338270453f8c9ef5b096a83cc |
| Mar 18 22:06:11 2018 | Merge branch master of https github.com twentytwoo zephyr | 69d1432d687fabcac5da28b76d0a23791aa85a41 |
| Mar 19 22:43:54 2018 | added larger enemy hitbox | aa7feb802f00e06d91ed9ee495a5f4c3fdc64bbb |
| Mar 20 01:29:19 2018 | killstreak ui code added | 9969d822887cbba652812384602b1c4ccc1efe05 |
| Mar 20 01:48:41 2018 | new song modified sound levels | c18246bd06e0eaee469739f79c35fdfe8b024aea |
| Mar 20 23:53:26 2018 | increased difficulty over time by spawning more enemies fixed bullet clipping shield hitting shield causing dmg increased pickup flashing time to a minimum of 1 second | b52dc08bcdadc1abe52388845d9d3c31046b4ad7 |

| Date | Commit message | URL |
|---|---|---|
| Mar 21 18:08:29 2018 | added timer difficulty scaler fixed score countint to go only from when player can move | acfb835693f4be6a87a88f39d493b8679625f14b |

**Table 1** git Commit log

## 9. Bibliography

### References


[1] I-PROGRAMMER. (2018). Sage - computer of the cold war, [Online]. Available: http://www.i-programmer.info/history/9-machines/441-age.html (visited on 03/27/2018).

[2] IBM. (2018). The first national air defense network, [Online]. Available: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/sage/ (visited on 03/27/2018).

[3] TESTED. (2018). How sci-fi propaganda art influenced the us and soviet space race, [Online]. Available: http://www.tested.com/art/43726-sci_fi-art-propaganda-across-cultures/ (visited on 03/27/2018).

[4] Wikipedia. (2018). List of cape canaveral and merritt island launch sites, [Online]. Available: https://en.wikipedia.org/wiki/List_of_Cape_Canaveral_and_Merritt_Island_launch_sites (visited on 03/27/2018).

[5] A. Games. (2017). Reassembly homepage, [Online]. Available: https://www.anisopteragames.com (visited on 12/05/2017).

[6] ——, (2017). Reassembly wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Reassembly_(video_game (visited on 03/26/2018).

[7] Wikipedia. (2018). Entity component system, [Online]. Available: https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system (visited on 03/06/2018).

[8] SSYGEN. (2018). Ecs vs yolo coding, [Online]. Available: https://github.com/SSYGEN/blog/issues/24 (visited on 03/06/2018).

[9] twentytwoo. (2017). Classes creation test code, [Online]. Available: https://gist.github.com/twentytwoo/38df41452b7ab047c316b0a8cdf34252 (visited on 12/06/2017).

[10] ——, (2017). Methods test code, [Online]. Available: https://gist.github.com/twentytwoo/7f23960802416bf175fb557fe3ee9781 (visited on 12/06/2017).

[11] ——, (2017). Inheritance test code, [Online]. Available: https://gist.github.com/twentytwoo/75419cc33364571cd2cfc45d506c6d71 (visited on 12/06/2017).

[12] Box2D. (2017). A 2d physics engine for games, [Online]. Available: http://box2d.org (visited on 12/06/2017).

[13] Rude. (2017). Love.physics, [Online]. Available: https://love2d.org/wiki/love.physics (visited on 12/06/2017).

[14] rxi. (2017). Flux, [Online]. Available: https://github.com/rxi/flux (visited on 12/06/2017).

[15] twentytwoo. (2017). Zephyr, [Online]. Available: https://github.com/twentytwoo/zephyr (visited on 12/05/2017).

[16] vrld. (2017). Hump camera, [Online]. Available: http://hump.readthedocs.io/en/latest/camera.html (visited on 12/05/2017).

[17] STALKER-X. (2017). Ssygen (adn), [Online]. Available: https://github.com/SSYGEN/STALKER-X (visited on 12/05/2017).

[18] Rude. (2017). Body:setbullet.

[19] Stencyl. (2017). Continuous collision detection (ccd), [Online]. Available: http://www.stencyl.com/help/view/continuous-collision-detection/ (visited on 12/07/2017).

[20] videah. (2018). Splashy, [Online]. Available: https://github.com/videah/splashy (visited on 03/07/2018).

[21] Leafo. (2018). Itch.io, [Online]. Available: https://en.wikipedia.org/wiki/Itch.io (visited on 03/27/2018).


### List of Figures



### List of Tables



[23] IBM. (2018). The first national air defense network, [Online]. Available: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/sage/ (visited on 03/27/2018).

[24] TESTED. (2018). How sci-fi propaganda art influenced the us and soviet space race, [Online]. Available: http://www.tested.com/art/43726-sci_fi-art-propaganda-across-cultures/ (visited on 03/27/2018).

## List of Listings

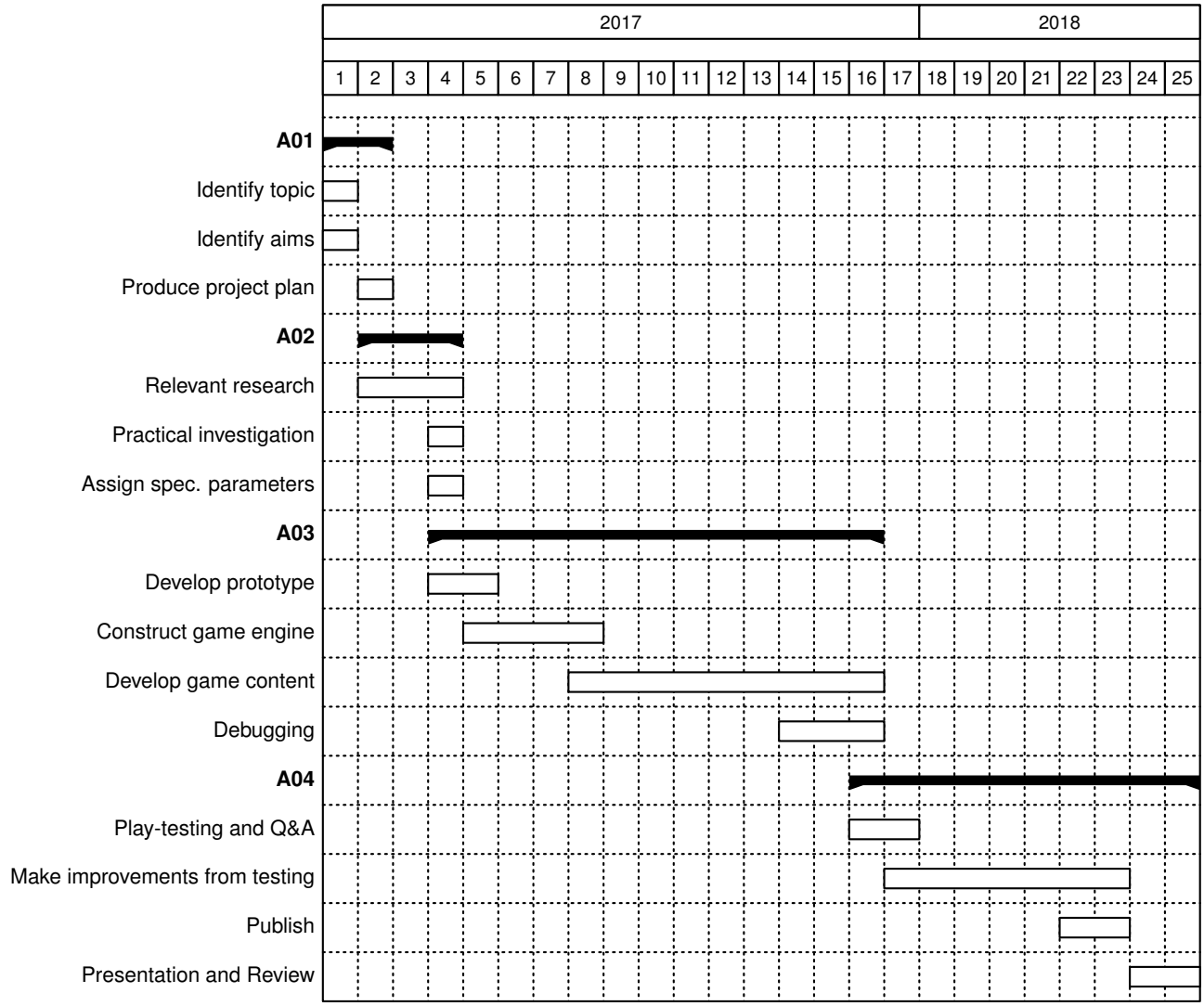



**Figure 46** Week by week project plan