Worcester Polytechnic Institute
Digital WPI

April 2014

Prota Evolution: A Genetic Platformer

Jennifer Renee Baulier
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Prota-Evolution: A Genetic Platformer

A Major Qualifying Project for WPI

Submitted by Jennifer Baulier 2014

Advised by Brian Moriarty

IMGD Professor of Practice

Abstract

Prota-Evolution is a game focusing around the idea of putting evolution into the hands of the player. The primary mechanic is a platformer, divided into levels, in which a mobile player agent faces enemies, obstacles, and a short time limit. Before attempting a level, the player gets a pool of ten creatures, each with four abilities. The top five creatures are ranked by the player, and her first choice attempts the level. If the creature fails, new creatures are formed from the top five creatures using genetic algorithms. Mutations may occur, adding new abilities, and these mutated creatures replace the unranked creatures in the player's pool. When a level is completed, an entirely new pool of creatures are generated to face the next level. Over multiple attempts, the player evolves a creature best able to survive the challenges of the game.

Contents

1. Project Overview

This project was taken on as a one-person technical game development MQP. The time duration for the project was one term, performed during my final term prior to graduation.

1.1 Project goals

General goals for the project included the following:

- A chance to try AI techniques in a game that I haven't programmed before.
- The creation of a game that is easy to demonstrate and works as a strong portfolio addition.
- An opportunity to see if a player-assisted AI algorithm can work as a fun gameplay mechanic.
- The opportunity for me to design a game entirely on my own and produce it from start to end.

Things I hoped to have in the game included:

- Creatures that undergo genetic mutation based on player input.
- Three platformer levels.
- A pool of primary abilities for the creatures, along with some mutations, including some rare mutations that are situationally useful.
- A timer visually incorporated into the level.
- Creatures and abilities that can be distinguished even by color-blind users.
- A clear and simple user interface.
- Some interesting obstacles such as water, magma, and enemies with different movement patterns.

Stretch goals:

- Touch pad support.
- Additional levels.

2. Design Process

Given the time restraints of the project, it was important that most of the design was ready before the project's official start.

During the term prior to the MQP, several game engines were contemplated, the basic game idea was finalized and mechanics were fleshed out. Ideas for the user interface were also developed. At the first project meeting, the final design was discussed and a first attempt at a user interface was shown. Upon approval, active development started immediately.

2.1 Inspiration

I was inspired by the idea of exploring little-used techniques of adaptability in gameplay. It intrigued me to find out whether such methods would result in fun and interesting gameplay.

2.2 Game Ideas

I originally proposed several game ideas based on the concept of adaptation. The ideas that were not chosen may still provide inspiration for future projects.

2.2.1. Modifying the type of challenge

My first idea was a roleplaying game where levels consist of a single battle with an enemy, followed by a decision. After beating this enemy you can choose to either train with it, befriend it, or shun it. If you befriend it then it joins your party. If you train with it, it gives some experience to the members of your party. Enemies will have special abilities that you will not be able to utilize if you do not befriend them, but your party will have fewer points in any given ability, or possibly less health/strength, if they do not train. After some number of levels/battles the player will have a boss rush of a somewhat stronger version of every enemy they shunned, possibly followed by a final boss. The fewer enemies the player shunned, the stronger the final boss is and the more powerful its abilities will be. This final boss will gain a huge boost if the player had not shunned any enemies along the way. If all enemies were shunned, then the player would only get the boss rush and would not face this final boss. The thing that makes this game intriguing is the idea that the hero character is sort of building his/her own fate and setting up what their final battle experience will be like. Based on the variety of possible play styles, players could have wildly different end games.

2.2.2. Genetic enemies

This game was somewhat similar to *Prota-Evolution* in that it involved genetic algorithms, but applied to the enemies, not the player. This game was a tactical roleplayer in which the player had to defend a base from waves of enemies. Each wave acts as a generation of creatures, and are ranked based on how effective they were in their attack, and how frequently the player used attacks that they are weak against. Subsequent waves consist of the results of the previous waves after undergoing a genetic adaptation algorithm.

2.2.3. Morphing creature

In this game, a player-controlled creature undergoes a series of battles in which they have points they can spend to cast spells. A spell can be an attack or a defense, and can be typed to one of four elements. During each battle, it is tracked how many times the creature attacks versus defends, and how many times each element is used. Based on these statistics, one of the creature's body parts that has not yet mutated would morph, and the creature would require fewer points to use that spell moving forward. Alternately, a player could instead choose to give their creature additional eyes, which allows it to learn a new magic ability. If the player makes it through eight battles they win and have a fully-morphed creature.

This idea was an extension of a game I made during a previous course. In that version, the player fought using a strange monster who gained arms, legs, tails, eyes, etc. after winning battles based on how the player decided to invest experience points.

2.3 Game engine

There were four game engines I considered using for this project: Dragonfly, Processing, Perlenspiel, and Flash Develop. In order to decide on an engine, I compiled a list of pros and cons for each.

2.3.1 Dragonfly

Pros

- A game written in C++ would be good for my portfolio.
- ASCII-based art and animated sprites is helpful since I'm not working with an artist.
- There is an online tutorial.
- I have code from my IMGD 3000 project to look at.
- There is structure in place to help with getting collision information, step updates, and camera scrolling.

Cons

- It has been a while since I've written anything with it, so substantial review would be needed.
- May be harder to make accessible to others to play after it is finished.

2.3.2. Processing

Pros

- Relatively easy to create art programmatically.
- The language is very easy to program and run.
- A new version is available that can be hosted on any website.
- Has a built-in game loop.
- Easy to present information to the user.

Cons

- No collision detection or movement via acceleration.
- Though quicker than drawing sprites, producing the art still could take a substantial amount of time.
- Scrolling backgrounds may be tricky.

2.3.3. Perlenspiel

Pros

- No art required at all.
- Easy to put on a website and make accessible.
- Easy to give text instructions, though space is limited.
- I am familiar with it and have several example games to look at.
- My advisor created it, and continues to update the engine, so if I run into any issues I can talk to him about them.

Cons

- May be hard to get the complex idea across to the player with just colored squares and individual text characters.

2.3.4. Flash Develop

Pros

- Easy to learn.
- Can be programmed in Windows through a VM.
- Easy to do physics, vector-based movement, collisions and camera scrolling
- Easy to design levels in a text document.
- Probably the easiest to get information across in, since actual art is used.
- Sound and music not hard to add.

Cons

- It has been a while since I've used it, so substantial review would be needed.
- Actual sprite art is required, which is time consuming to produce, and I am not an art student

Ultimately I decided on Perlenspiel. A primary reason is that there is no need for art assets whatsoever. This allowed my entire focus to be on the programming, which should be the case for an IMGD tech MQP. This was also important given that I am on a tight time limit, and creating art is a lengthy process. Another major reason is that my project advisor created Perlenspiel, and thus would be able to help me if I did not know how to make something happen in the engine. Furthermore, if we could not find a way to efficiently do something I needed, there changes could be made to the engine.

My final reason for settling on Perlenspiel was the ability to host games made with it on any website. Having a project easily available to show others is very important, especially given that I am trying to build up a portfolio. I had previously done my CS MQP in Osaka, Japan. That project was interesting, but it is not currently possible to directly show it to other people at this time. I did not want the same thing to happen at the end of this project.

2.4 User Interface Choices

There were several considerations to be taken into account when designing the user interface. One was that every important object would need several ways of recognizing it, to make things easier for both color-blind users and users who may have trouble seeing small characters on a computer screen.. For example, creatures have both colors and letters indicating which creature they are. The abilities of the creature, which are very important, have colors, symbols, and written descriptions that display when moused over. Abilities are seen by mousing over the individual creature, or clicking the Show All Abilities button. The bar displaying the time during the level changes color, and gets shorter near the end of the time limit. The color of the timer bar is also mirrored by the color of the background outside of the game. Status line messages indicate when time is running low. Enemies, platforms, and other obstacles all have both a symbol and a character to help them stand out from the background, and all platforms use checkmarks to show that they are safe.

I also decided to avoid having a lot of in-game text. While some use of text was inevitable, such as the descriptions of abilities, too much text could break immersion. Another reason for this decision was a desire to make the game understandable on two levels: a simple level for standard players, and a more complex level for players who want to understand the details of what is going on. Deeper details could be provided on the game's host web site without cluttering the game screen itself.

Pick your top 5 creatures by dragging them.

S T A R T		**Y O U R**			**E S C A P E !**				

S H O W		**A L L**		**A B I L I T I E S**					

1	2	3	4	5					
A	B	C	D	E	F	G	H	I	J
ʎ	ʎ	ʉ	ʉ	>	ʎ	ʎ	ƛ	ʎ	ƛ
†	ʉ	^	>	†	•	~	ʉ	8	†
ʎ	†	^	ʉ	ʉ	†	ʎ	†	ʎ	^
ʉ	ʎ	>	>	†	ʎ	ʎ	ʉ	ʎ	>

The creature menu, with Show All Abilities turned on.

3. Game Description

In *Prota-Evolution*, the player navigates creatures through various environments using platforming skills. At the start of a level, however, the creatures available may not have the necessary skill set or speed to survive in the environment long enough to make it to safety. The player's second job is to figure out what set traits should be considered most fit in the current environment such that the population will contain suitable creatures.

3.1 Overview

The player starts with a pool of ten creatures, each with four abilities. The player can drag creatures in order to rank their top five. Their top choice acts as their playable character for the next attempt at the stage. If the creature does not succeed genetic algorithm is used to create five new creatures from the parents in the top five positions. These children replace the unranked creatures. This continues until the player wins the stage. At that point and entirely new pool of creatures is generated.

3.2 Platforming controls

For many actions there are multiple buttons that can be used so that players can find a control scheme that is comfortable for them.

Action	Keys
Jump	space bar
Move left	Left arrow, d
Move right	Right arrow, a
Attack	Down arrow, s, c
Special one-off attack (unlocked by mutation)	w, x

3.3 Player stats

The playable creatures have five main stats that affect gameplay. The creatures begin with five points in each stat prior to being modified.

3.3.1 Strength

The first stat is strength, which manifests in two ways. A creature with higher strength will do more damage when it attacks, and its attack will also reach enemies at a greater distance. The damage done to an enemy in one standard attack is equal to $5 + floor(1.5 * strength)$. The attack hits some number of blocks in both directions, and this number is determined by $floor(strength/5)$.

3.3.2 Health

The second stat is health, which simply affects how much damage the creature can take. This is represented by a number called hit points. When an enemy deals damage the player loses hit points, and if these go to zero the attempt at the level ends. The number of hit points the creature starts with is equal to 20 + (8 * floor(health/3)).

3.3.3 Speed

The third stat is speed, which in term of mechanics determines how quickly keyboard input is accepted. As a result an increase in speed will have effect on jumping, walking, and attacking. The base time in sixtieths of a second between keyboard input being accepted is equal to 25 - 4 * floor(speed/3), and this cannot go below 1. There is a times when this speed is modified again though, and that will be explained later.

3.3.4 Vertical jump

The last two stats are attributes of jumping. The first is vertical jump, which determines how many times the creature will attempt to move upward before gravity makes them fall again. That said, if the creature hits something above them, other than the top of the stage, before the upward part of their jump is done, they will start to fall anyway. The number of times the creature will try to move upward if they do not hit anything is equal to 2 + floor(vertical jump/3).

3.3.5. Horizontal jump

The other jump-related stat is horizontal jump, which is modified in two ways. A creature with a higher horizontal jump will take longer to reach the maximum number of upward movements, allowing them more time to move around in air. The time each step of the upward portion of a jump takes is equal to 10 * floor(horizontal jump / 5). The player will also have an increased keyboard input time while on the upward part of their jump, based on the creature's horizontal jump stat, allowing in-air movement to increase even further. The new keyboard input time during this phase is equal to 3 + the old keyboard input time - floor(horizontal jump/2), and this cannot go below one.

Given that jumping is so important in a platforming game, it seemed reasonable to make the player pay attention to two aspects of jump instead of giving them a single attribute that might prove to be overpowered.

3.4 Abilities

In the first generation, creatures take abilities from a base pool of possibilities. There are five abilities in the base pool. Each raises one of the five basic stats by six points.

In later generations, creatures not only inherit abilities from their parents, but also have a chance of mutating to gain new abilities. There are two types of mutations: a more common sort that pulls from a secondary ability pool, and special mutations.

3.4.1 Secondary abilities

The secondary pool contains three abilities. One is called **General Jump,** which raises both the horizontal and vertical jump stats by three points. Another is **Full Fighter,** which adds three points to the Health and Strength stats. The third is **Well-Rounded**, which adds two points to *all* base stats. Whether these two points are enough to affect game play depends on what the creatures stats were previously, but generally it will make a difference for at least some of the stats.

3.4.2. Special mutations

Special mutations can only happen after a certain number of normal mutations have occurred during a level. These mutations are powerful and useful, but only in rare situations.

The first special mutation is **Fire Resistance**. This causes the player to take half damage from lava and lava enemies. It is very helpful if someone was having trouble getting through stage two.

The second special mutation is **Night Vision**, which lightens the color of the sky, allowing the creature to see in the dark. This has an effect in all levels, but is only useful in stage three, in which the sky is pitch black with dark enemies that sometimes blink to show their locations.

The third special mutation, **Thick Skin**, halves damage from neon enemies. All enemies in stage three are considered to be neon.

The final special mutation, **One-Off Attack**, is useful anywhere, but can only be used once during an attempt at a level. It allows the player to perform a single attack that covers three times the usual distance and does two times the damage. Normal attacks can still be done an infinite number of times.

3.4.3. Symbols

Each ability and mutation is represented by a unique color and symbol.

Ability	Indicator
Horizontal Jump	
Vertical Jump	
Speed	
Strength	
Health	
General Jump	
Full Fighter	
Well Rounded	
Fire Resistance	
Night Vision	
Thick Skin	
One-Off Attack	

3.5 Obstacles and enemies

In the first level, there are five types of enemies and two types of obstacles. Enemies differ in the amount of damage they deal, the amount of damage they can take, and the way they move.

Enemies do damage by touching the player. If the player walks into or jumps onto an enemy, even if it is stationary, the player will take damage.

The levels also have a time limit acting as an additional challenge the player must consider. The time limit is indicated by a timer bar that turns a reddish color and then the color of that stage's sky, so it looks as if the bar is ticking away. The color of the left-most square of the bar is mirrored by the color of the background outside of the game screen. Messages alert the player when half their time is remaining, and warn that time is almost up when only ten squares of the timer bar are still visible. The status messages also indicate when damage is taken by saying "Damage take! HP left: #," where # is the player's remaining hit points.
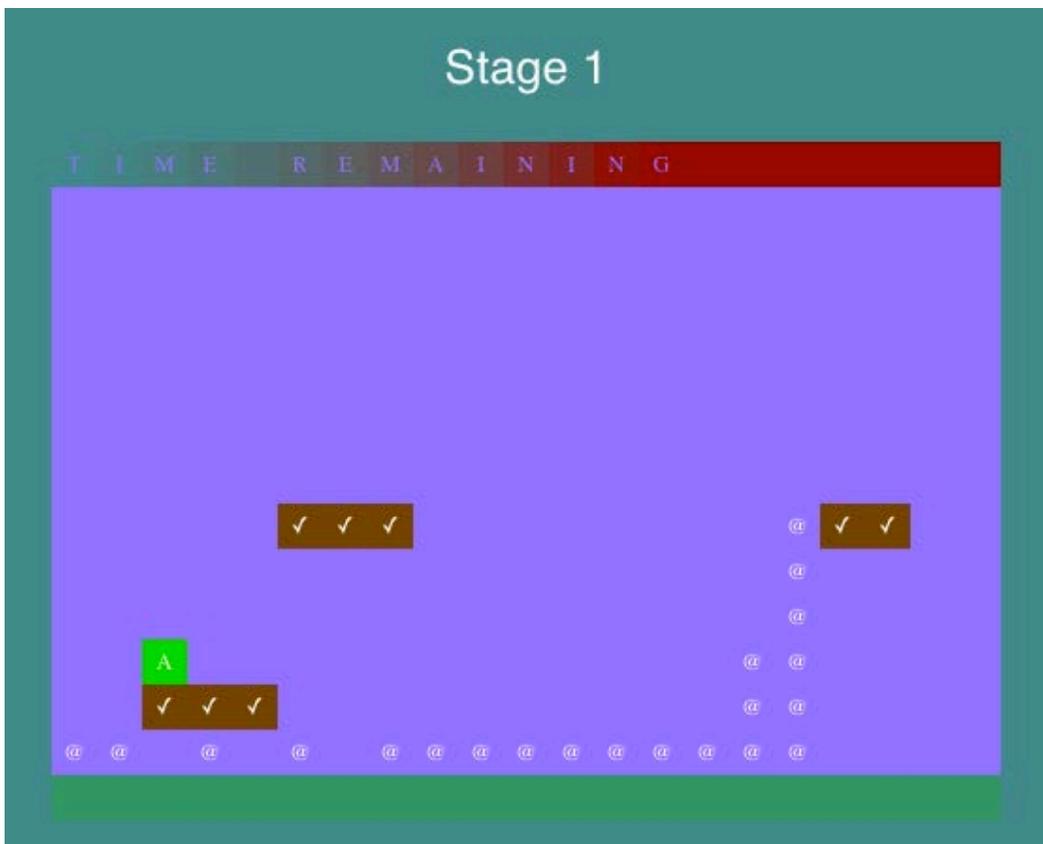
The Stage 1 timer on the top of the game window, approaching the halfway point.

3.5.1 Level 1 obstacles and enemies

The easiest enemies are represented by white "at" signs (@), and are either stationary or pacing. Lines of stationary enemies are sometimes used as barriers that cause player damage if touched. Pacing enemies move until they hit something, walk to the end of a ledge, or reach the end of the currently visible screen, and then they switch direction.

A second enemy type is very similar to first. These are represented by red percent signs (%), and are stronger, have more health, and move faster when pacing.

A third enemy type, represented by brown Ws, starts in the air and moves up and down a provided number of spaces from their central starting position.

Cyan "X" enemies fly in a circular pattern from their starting position, leaving a three-by-three square of tiles untouched in the center. In level one, a platform is put in the overlapping space in the center of a flock of these creatures' movement circles. It is possible to set what part of the creature's circular path it is starting on.

The final type of enemy is a steel blue ampersand (&) that does nothing except block paths. This creature can take a lot of damage, and does a lot of damage if the player touches it. If the player does not have a lot of strength or speed, these enemies may make it hard to avoid hitting the time limit.

Obstacles in the first level include standard platforms and moving platforms. Standard platforms can be floating at any height. Moving platforms in this level move up and down a certain distance from a start position, similar to the brown W enemies. When the player's creature jumps on these moving platforms, they stay at the height they are at until the player moves off of them, then resume their standard movement.

3.5.2. Level 2 obstacles and enemies

The second level is themed around lava, and contains variations on many of the enemies found on the first level. These variations are lava-colored (orange), and are considered to be lava enemies, subject to the fire resistance mutation. These enemies act like their first-level counterparts enemies but give more damage and have more health.

Throughout this level, there are lava-colored squares that are harmful to creatures. A special enemy for this level is a volcano that periodically shoots streams of lava upward. This is considered an enemy, because the player creature can attack it, and touching it will damage the player. These take a fairly high amount of damage. If the player's creature jumps on the stream of lava, it will be pushed upwards to the top of the stream, taking damage along the way. If the stream of lava finishes, and the

creature is still on top, the creature falls as the lava goes back down. The volcano itself is not considered a lava enemy.
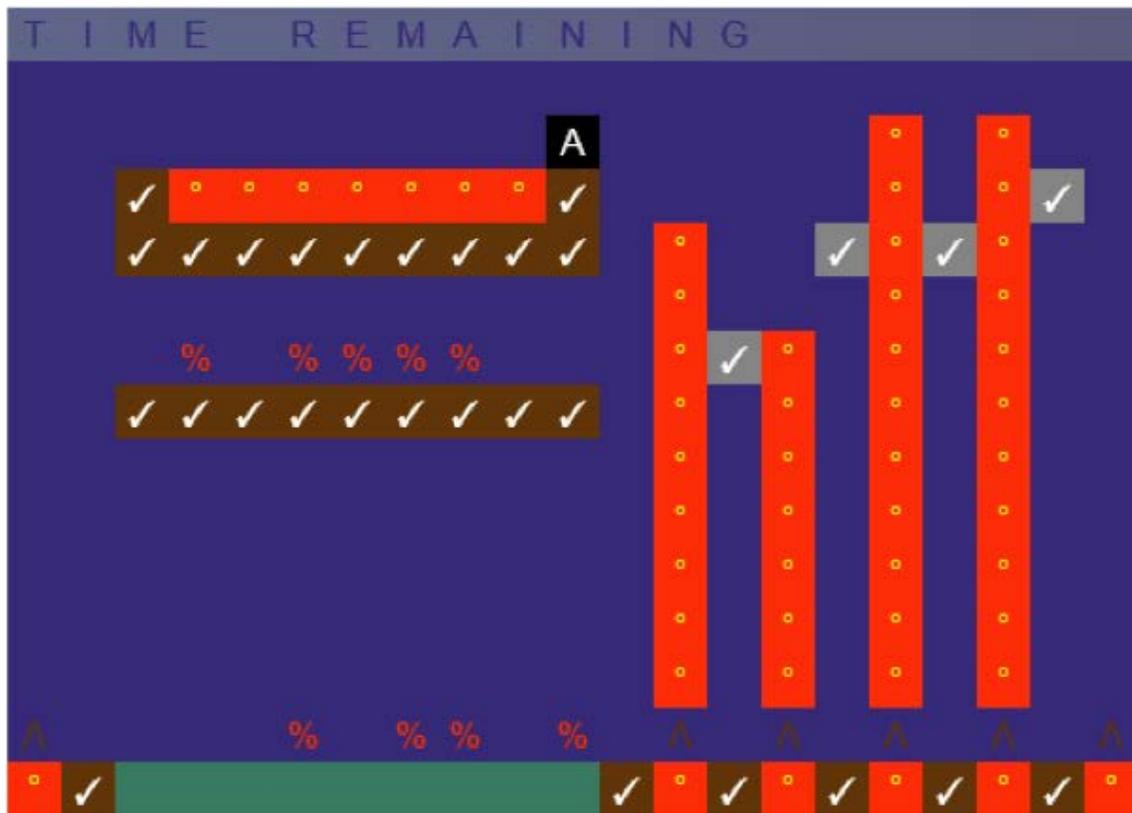
A screenshot of Stage 2.

3.5.3. Level 3 obstacles and enemies

The third level is themed around darkness. It contains enemies that switch between neon colors and black, causing them to blend into the background half of the time. Players can still interact with them when they are colored black. All enemies in this stage are considered neon enemies, subject to the Thick Skin mutation.

Most neon enemies are variations of the enemies in the previous two levels. The W enemies do not blink, but this stage's percent sign (%) enemies do. A new enemy to this stage is shaped like an eight (8). Meant to resemble fireflies, this stationary enemy blinks neon-yellow and black. It is often floating in the air, forcing the player have to pay attention when jumping.

Some platforms on this level also blink. Stationary platforms are neon-green with black checkmarks, and may or may not blink. Moving platforms move exactly as they did in the previous stages, but also blink neon-blue and black.

A final obstacle unique to this level is a set of pink arrows (<). These do not blink. If the player's creature touches an arrow, it gets moved past the arrow in the direction the arrow is pointing.

A screenshot of stage 3.

3.5.4 Tile symbols

Name	Picture	Hit points	Damage it deals	Other
Basic enemy		15	5	Paces (stage 1)
Red ant		20	10	Paces (stage 1)
Brown bird		15	5	Vertical flying (stage 1)
Cyan bee		15	15	Flies in a circle (stage 1)
Steel blue sentry		75	25	Stationary (stage 1)
Standard platform		Infinite	None	Stationary (stage 1 & 2)
Moving platform		Infinite	None	Vertical movement, stops when jumped on (stage 1 & 2)
Stage 1 grass		Infinite	None	
Stage 1 sky		Infinite	None	
End goal		Infinite	None	Move into to beat the stage
Timer start 1		Infinite	None	The start color for the timer bar in stage 1
Timer half 1		Infinite	None	The half-way color for the timer bar in stage 1
Lava ant		20	14	Paces (stage 2)
Volcano		50	2	Periodically shoots lava upward (stage 2)
Lava bird		20	12	Vertical flying (stage 2)
Lava bee		18	26	Flies in a circle (stage 2)
Lava		Infinite	4	(Stage 2)
Stage 2 grass		Infinite	None	
Stage 2 sky		Infinite	None	
Timer start 2		Infinite	None	The start color for the timer bar in stage 2

| Timer half 2		Infinite	None	The half-way color for the timer bar in stage 2
Neon ant		20	14	Paces & blinks (stage 3)
Neon bird		20	12	Flies in a circle, but does not blink (stage 3)
Neon Firefly		20	8	Stationary, but blinks (stage 3)
Chemical spike		Infinite	2	(Stage 3)
Neon arrow		Infinite	None	Player shoved in that direction if they walk on it (stage 3)
Neon platform		Infinite	None	Stationary & may or may not blink (stage 3)
Neon moving platform		Infinite	None	Vertical movement, stops when jumped on, & blinks whether moving or not (stage 3)
Stage 3 grass		Infinite	None	
Stage 3 sky		Infinite	None	
Timer start 3				The start color for the timer bar in stage 3
Timer half 3				The half-way color for the timer bar in stage 2

3.6 The Genetic Algorithm

Genetic algorithms are an artificial intelligence technique that is inspired by evolution. The algorithm starts by establishing a starting set of K creatures or objects. These creatures have some set N of attributes representing varying traits, similar to chromosomes in DNA. This starting pool is the zero generation, and can be chosen randomly or from some previously obtained starting point.

To start the genetic process, the creatures in the zero generation are ranked based on some fitness function that must be defined. The fitness function can simply be an algorithm, or can involve putting the creatures through some test and ranking the results.
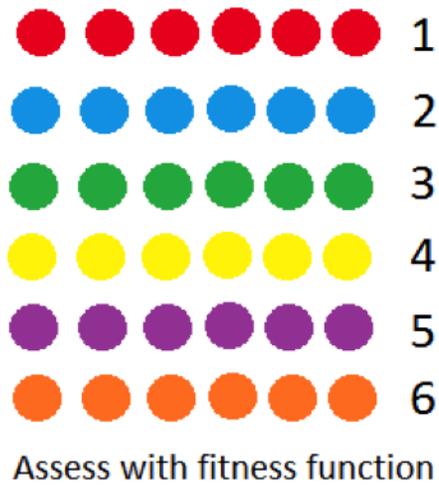

Step 1

Once every creature in the generation has a ranking, pairs are chosen. A given creature can be in multiple pairs. The members of each pair are chosen randomly, but the likelihood of a creature being chosen is based on its fitness ranking. Normal K/2 pairs are formed.
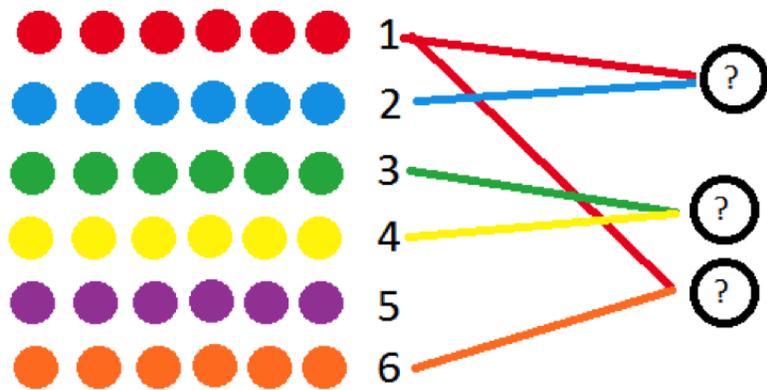
Select random pairs of parents weighted by fitness

Step 2

After pairs are formed, attributes are taken from each member of the pair. Normally an equal number is taken from each, but there is randomness in which attributes come from which parent creature. After a full set of N attributes is chosen, it is randomly decided whether any of those attributes will undergo mutation. In this case, mutation can be any sort of change from the attribute that the creature would have received.
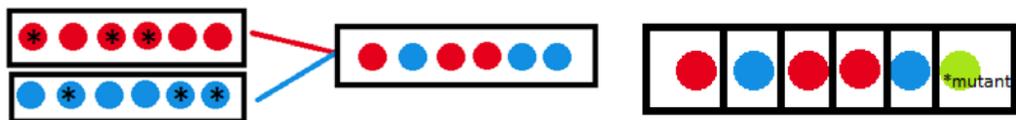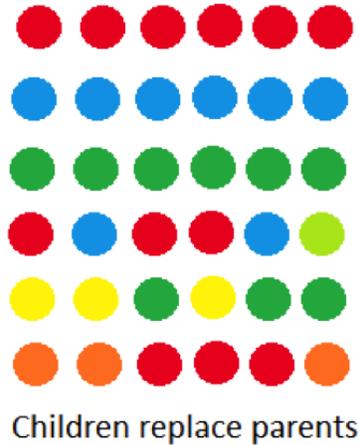

Randomly select attributes from each parent   Check attributes for mutations

Steps 3 and 4

Finally after all child creatures have been generated, they replace the parents with the lowest fitness in the pool of creatures. These children are now considered generation 1.

The genetic process can continue for any number of generations.

Children replace parents

Final step in forming a new generation.

In *Prota-Evolution*, the zero generation consists of ten randomly-generated creatures. Each creature has four abilities randomly chosen from the base pool of abilities. These abilities fill one of four positions in an array. In the menu, this array is graphically displayed vertically below the creature if the player mouses over it or clicks the Show All Abilities button.

The fitness scores of the creatures are chosen based on the player's personal rankings. In this case, the player ranks their top five creatures, and only those five are considered as candidates for "breeding." Using these five creatures, five pairs of parents are chosen at random. The likelihood of each parent being in a given pair is weighted by the one-to-five ranking the player provided for it.

Once a pair is chosen, an ability is decided upon for each position in the child's ability array. If no parent has provided two abilities, then it is completely random from which parent's array the ability in that position will be taken. If one parent has already provided two abilities, that ability is taken from the parent who has not done so yet. In this way, it is completely random which parent provides which ability.

It is important to note that abilities stay in the same position they were in when they were in the parents' array, because that means two traits that are only in the same position cannot both be inherited by the child.

Once an ability is chosen to fill a given spot in the child's ability array, if the child has not yet undergone any mutations, it is randomly determined whether that ability will mutate into a different ability before the ability is added to the array. Mutations can either result in the creature getting abilities from the secondary pool or the special pool, but not from the base pool. Once it is determined that a mutation will occur, an ability is chosen at random from within the correct pool, though an abilities are not allowed to mutate into the same ability.

Because only allow one mutation is allowed, and mutation-checking runs from the beginning of the array to the end of the array, the chance of a mutation increases slightly for abilities farther down the array.

A special mutations can occur only if ten standard mutations have occurred since the start of the given stage. Special mutations are checked first, and if that does not occur than the ability is checked for standard mutation. Mutations are determined by randomly generating a number between one and some maximum value, inclusive. If the maximum value is generated, mutation occurs. For special mutations, the maximum value starts at eighty, and then decreases by five at each position in the array. The maximum value for standard mutations starts at 50 and again decreases by five for each position in the array.

When all children have been generated, the unranked parents in the previous generation are replaced by the newly-generated children. This continues for as many times as the player attempts to complete a given level. If the player wins the level, an entirely new pool of creatures is generated, and the number of mutations is reset so that ten standard mutations must occur before special mutations can start.

4. Evolution in Gameplay

Finding out an interesting way of having some feature of a game undergo evolution was one of my primary goals going into this project. I also wanted to see if giving the player some control over this evolution made for an interesting gameplay mechanic.

4.1 What evolution adds to this game

I feel that the aspect of evolution added a lot to *Prota-Evolution*. First, it introduced a form of exploration that the player had to undergo. The player may not know exactly how a given ability will effect a creature, or what abilities will work best in a new environment. The player must take a guess and then actually put the creature out into the environment and see what results. From there they can revise their guesses, hopefully improving their idea of what is most fit over time.

A second thing evolution adds is a novel way of thinking about the game. One of my play testers described the experience as a counter-intuitive sort of gameplay, where to start you may not have something that is good enough to beat the level, but over time after several losses you work towards creating things that are. This tester considered this style of play rather interesting.

4.2 Future potential

Though evolutionary techniques are already seen in games, there are only a few common manifestations, such as branching stories, characters that change based on moral decisions, creatures that change in appearance as the player gains experience, or levels that programmatically modify to adjust difficulty. I believe there is a potential for much greater variety.

When thinking about new ways games could employ evolutionary adaptation, there are two main things to consider: what adapts in a game, and why it adapts. Within these two areas there is a lot of room for experimentation.

One possibility is the player's end goal. This is the sort of thing that can add a lot of variety to successive gameplay, and might encourage players to pay more attention to different play factors, since the goal they are working towards may change.

An opportunity that I feel has a lot of promise is finding new methods of adapting the play style of AI opponents. This could have a huge impact on replayability, and even within a single playthrough, it might hold a player's attention for much longer.

Other aspects that might evolve including the setting and ambiance of a game, or even the basic mechanics. This could be taken to the extreme where the actual genre of the game changes, although a lot of thought would have to be put into making the gameplay remain clear to the player.

Even if a player knows what is adapting, they must then figure out when and why it will adapt. Different factors affecting the evolution of the same feature could create very different experiences. One factor that could be used to determine adaptations is the player's play style. Watching how a player goes about a game may tell us something about what they find interesting or challenging. For a teaching game, the player's learning style could be taken into consideration by determining if the player learns better through visual, auditory, or other means. Lessons could then be adapted based on what might prove most helpful. If a game was able access a player's outside interests or demographics, ways might be found to bring content that may appeal to them into the game. This is something I have not yet seen done with gaming, but see utilized increasingly with other media.

Using tools such as a heart rate tracker, the player's current mood or emotion might be gauged. It would be interesting to see the effects of using these parameters to change things in a game. This would be a good factor to pair with something such as game world and ambiance, but also could affect other things such as the pacing of the game.

A more obvious measure the player's skills and weaknesses within the game. The game can either try to help them improve things that they have already proven to have some proficiency at, or it can play at their weaknesses in order to force them to become more well-rounded. Also considering specific strengths and weaknesses might be a way to better fine-tune adaptive difficulty in games.

Finally, a game could pay attention to the player's preferences within the game. This can be done with eye tracking devices, and can be used to tell if a player has a strong interest in a given setting, plot-point, character, or challenge type. Based on this data, more content relevant to the specific things the player finds appealing could be added, or information related to those things could be delayed to add suspense.

5. Code Features

While the genetic algorithm was probably the most thing from a coding stand point, the scrolling map in my game along with some of my major classes are also worth consideration.

5.1 Scrolling Maps

In order to set up scrolling maps I first represented the entire map of a level as a two-dimensional array. I kept track of the current edges that are be displayed on screen and made a function that takes a map and a direction. To start off the direction is zero, but during every other call to the function the direction will either be negative one or one. This indicates how to shift what is currently displayed. As a note the map only scrolls left and right, not up and down. First the direction is used to find the new edges of the map, and hen the function parses through every element of the array that are within those edges.

Elements are read one at a time and every element is a tuple containing all necessary information. In many cases the tuple will only contain a single integer indicated the type of the object, but even if the tuple contains more information the type will always be first. Things such as sky, grass, and stationary non-blinking platforms only contain the type even though the color is different by stage for the first two. Things that need additional information include enemies, moving platforms, and lava that comes out of volcanoes. For these objects the first information following the type will be the things needed to create the class for that object. Anything longer than that are optional fields. Enemies and moving platforms use these fields for their specific move function followed by an array that gets stored in an optional field that both of those classes have. The optional field will often be used by the specific move functions. Also moving objects have a timer field that starts out null, but often this timer will be started by the move function itself. Lava from volcanoes, as mentioned is treated as an object that is sometimes on screen and sometimes not, but the space is still tracked where it would be. Volcanoes keep an array of these and toggle them on when they want them to be showing.

When a player moves in either direction past the midway point of the map a call to fill map will be made indicating a shift is needed if there is more of the map that is not displayed in that direction. Given that creatures and platforms move out of their initial position, and they do this while the map is also shifting, I need to pay careful attention to them. When their initial positions are parsed these location still need to filled with sky color even though the creature is likely still there. Then after the basic parts of the map are filled in existing moving objects must be pushed one spot in the direction opposite of the direction the map moved in and new moving objects must be added. Also, given that the enemy may be pushed off screen it is important to check whether they should actually be drawn after the map scrolls and inside of their movement function.

5.2 Major Classes

One of the most important classes in my code was the game stage class. This contained a lot of timers and information important to the current state of the world. My second most important class was my Creature class which was used for every generated creature in the pool, though it did contain a little bit of information that was only needed by the player character. It contained the creature's five base stats as well as its array of abilities. It also contained its global position, the distance of its attack and Boolean for each of the four special abilities gained through special mutation.

The enemy class contains its health, damage, color, and glyph. These four features are passed in on creation. Additionally there is an important field that indicates whether the enemy is dead. The enemy also has fields that it may or may not utilize for its move function, move timer, initial position, direction and the field called extra that its move function may make use of. The class for moving platforms is similar though needs fewer fields. Also temp items is the class used for things like the lava that comes out of volcanoes. As mentioned these are treated like they are sometimes on screen and sometimes not, as a result when they are on screen they are drawn onto the map at the same time as the moving enemies.

Ability is the final important class and this contains four fields all of which are passed in on creation. The first is a number indicating the ability. This is used for several things, one being looking up what symbol represents it. The second field is a function which tells what should be called once the player character is decided, such that it actually gains the intended ability. Most abilities use a function called addStat which just adds a given amount to the needed stats based on how many stats it is adding to. The amount for each number of stats is set as a macro at the top of the code. The third field is a list of all stats effected. For the abilities I ended making addStat was the only function that utilized it. The last field is the description. This contains a string indicating what to say when the player scrolls over that ability in the menu. The strings are retrieved using a function as soon as the number of the ability is decided.

6. Testing

With every game that is made testing is a crucial part of the process. Games need to go through many iterations, and balancing values is often a very tricky problem to handle. Though the creator will notice some problems through playing, it is good to have a large amount of the testing done by other people who are not yet familiar with the game.

6.1 Initial balancing

Before I had official play testing I had to make an initial guess at what values to use for stats and other properties. I played around with different numbers and so how it affected the creature's movement. Then once I got in the genetic algorithm I created creatures with a different numbers of a given ability and saw how movement changed. Using this method I modified numbers I was using and got it to a point where I at least felt comfortable with it.

6.2 Playtesting

The game underwent playtesting with six different players, excluding me, and each time new things to consider were observed. Play testing was done by having the user run the game in Chrome on their own machines while I observe them on Skype using both voice chat and screen sharing. I took notes as I watched and made modifications between each play through. Though users will remain anonymous, interesting findings from each playtest can be seen bellow, along with decisions that were made as a result of testing.

6.2.1 Playtest 1

This tester really did not like having to scroll over the narrow column of abilities. He thought the fact that they disappeared if he moved to the side was frustrating and having to scroll over a creature to see its abilities was a waste of time. He wanted to see a table of all abilities at all times. I thought this might be confusing to look at and tell what corresponds to what creature, but it lead to me reconsidering my display method. He also did not like c to attack because he was using a/d instead of the arrow keys making c harder to use. As a result of this s was added as a possible key to attack with. Also, I have a special ability that is triggered with x, and w was added as an option to additionally work better for players using a/d. A major thing he noticed was that after the first level there was not enough use for pure fighting stat. A big part of this problem was a large lava pool at the end of level two. This was modified to be smaller, and more enemies were put in its place. In the other two stages It seemed that fighting was still important, even though I chose to make having good jump stats completely necessary in the final stage. Another thing he requested were temporary frames of invulnerability after taking damage. I do

prevent the player from taking damage again if neither it nor the enemy has moved since damage was taken, but a general time of invulnerability has not been implemented. This is something that I feel would need a lot of testing after being put in the game. He also helped me notice that enemy descriptions in special abilities, such as "glowing enemies", were not obvious to the player.

6.2.2. Playtest 2

This playtester at one point ended up losing a base ability he felt he needed and had to hope for mutations that affected that stat. As a result he requested a gene pool reset button. This requested was ultimately decided against though because I felt the player easily could abuse it. In the case of the first level it is pretty easy for the player to reset their gene pool until they get something they may be able to beat the level with. Since level one is supposed to be fairly simple and easy to begin with this is acceptable. I did not want to give the player the ability to do that every level and wanted to ensure that breading mechanic was a critical feature of the game. Even if things were needed that could only be achieved through breeding I felt it allowed the player to cheat their way into too good of a position to begin with. His playtest also helped me realize that base health should be somewhat higher, which was adjusted before the next test. Also he wanted sound to make it clearer when the player hit an enemy and when an enemy hit the player.

6.2.3. Playtest 3

This playtester, like tester 1, did not like having to scroll over each creature in order to see its abilities. He however provided a solution that I thought would be very useful. He suggested the implementation of a show all button such that the menu could toggle between the standard mode of display and the mode that shows all stats in a grid. I chose to have it start with the scrolling method given I feel it is more intuitive the first time the player sees the game, but the grid is more helpful once a player knows how the game works. This play tester felt that down should also be an attack button for arrow users to parallel the s-attack button that WASD users are likely pressing. This was also added, though c was kept because I liked having my attack next to my jump. An unexpected point was also brought up relative to the "well rounded" mutation. He felt that if a player got a creature with all four slots filled by well rounded it would be the best possible creature in most situations, but if the player had any fewer than all four of this ability it was not over powered. I asked him if he felt I should weaken it and he did not think this was a good idea. The logic behind this was that the ability is not that strong in most cases, and the one time when it is the best can only occur if the player has lost the level a huge number of times. He felt that this was actually a good thing since it acts as difficulty adjustment, and is something the player would have specifically worked towards. I thought this was pretty interesting.

6.2.4. Playtest 4

This player personally did not like the fact that the levels had a time limit. He felt that the challenge added wasn't necessarily beneficial to game. I felt that it made speed more meaningful in some places and kept it in. None of the other play testers seemed to mind it, though maybe it's something to consider if further testing was done. He also felt that attacks should not only go left and right but also up and down. I considered this for a bit realized that it could lead to odd situations such as the case where the player is quickly attacking downward while landing on something in the middle of a jump. I also felt that it would break a lot of the level design I had and in general make it harder to make platforming difficult. Another thing this player said was that they did not like how the jumps moved slower going up and faster going down. They wanted me to split the time evenly between the two. I felt that this should at least be attempted to see what the result would be. I was going to have the next player test with this version of jump. On implementation I played around a bit and quickly felt that I did not like the effect that it had on the feel of gameplay. It made the jump as a whole feel slow and kind of frustrating, and I liked having the falling action have a consistent pace. I ended up reverting it before doing the next playtest. No other tester mentioned an opinion on this aspect of the jump mechanic.

6.2.5. Playtest 5

This player's was confused about what was and was not safe to jump on, but I told him that the ultimate plan was to make a website that explained the specific details. I mentioned that everything that is safe to jump on has a checkmark on it except for the grass itself. He also was not sure why he died the first time because he did not notice the messages on top of the screen that told him when he took damage. Ultimately, if time provided, it would be useful to add sound and some other indicators to make it more obvious when the player takes damage. He also thought that the timer bar was not obvious if you did not know to look up there. This was part of what later lead to me having the out of game background change color with the timer. This effect was originally suggested by professor Moriarty, though I unfortunately did not think to use that solution until after my last playtest. As a result of this I cannot really say if adding this was helpful or distracting.

6.2.6. Playtest 6

This tester pointed out that the wording of certain status messages could be misleading and helped me rewrite a few of them to be clearer. He did not like using s, c, or down arrow to attack. His explanation as to why was the fact that he was using a/d to move and did not want to hit s with his middle finger in case he accidently tapped his movement keys. In general he felt that configuration was too cramped. He said if he was able to map his own keys he would leave movement at a/d and would put attack on j so his right hand could control it. Then he would be able to have his two thumbs on jump. I did

not add this option given I already added two new attack buttons for other play tester's keyboard layout preferences. Given that this still was not key choices for people, if time permitted having a menu where players can map their own keys is likely a good idea. A major thing this player talked about was the desire for obvious feedback as to whether or not he damaged a creature. Even though the arrow blinked where the creature was he felt this did not make it clear enough that he hit something. This is another case where sound would be useful. He also did not think it was clear that your attack arrows do not hold the enemy off if it is marching towards you. I was not so sure what to do about it. If I made it automatically move away from you when you were hitting than a player with a large attack distance would never get hit. It would be useful if I found a way to make this clearer but I have not thought of a solution yet.

7. Post Mortem

Whenever a game is finished, it is important to reflect on the process. I feel I was very fortunate to have the opportunity to do this project alone in the course of a single term without any other classes. I think it was a good learning experience.

7.1 Goals completed

I am fairly confident that I reached all of the general goals mentioned above. I definitely got the chance to try out an AI technique that I have not yet programmed. This was a fun experience. It turned out to be pretty simple to program a basic genetic algorithm, but it still was a fun experience. It was very nice seeing results that seems to show that the algorithm was working how I intended it to. The project being made in Perlenspiel definitely made the game easy to show, though I may work on an instructional webpage before fully using this as a portfolio piece.

In terms of seeing whether or not an AI algorithm could become a fun gameplay mechanic, I think I was able to see that it would. My playtesters seemed to think the concept was cool after playing, and after my presentation a few people seemed interested in the concept. To really see how fun this sort of thing could be though I am guessing farther testing would be needed. The final general go of designing and creating a game entirely on my own was fully met, given that this idea was my original concept and I saw it to completion.

I ended up being able to include every point on my list of features I wanted included in the game, though I did not get to my two stretch goals. One of my stretch goals was touch capability. I would need to figure out a control scheme that works for that given the player jumps, attacks, and moves. Given that this is a difficult design challenge of its own I am okay with not having met that goal. My secondary

stretch goal was just additional levels. I am rather happy with the three I made and do not feel more were needed to get the idea across. I also did not have many more level ideas at the moment and likely would have needed to ask for help coming up with something interesting.

7.2 What went well

I think the implementation of the genetic algorithm went very well, and so did the design of the main menu. I also think I was able to stick to goals to get done by certain times pretty well. The fact that I was working on my own was likely helpful in that area. In general the fact that this game was designed entirely on my own and actually made through code completion and testing is something fairly meaningful to me.

7.3 What I would change

I would like to have done some more playtesting and maybe found someone who actually knows more about game balance to help me polish my numbers a bit better. That said my values seemed okay to me but it's hard to really tell with only six play tests, especially given I do not really have experience balancing stats.

Another thing I would possibly change would be my level design. If I knew how too I may have attempted to make each level a bit more of a puzzle, or somehow further make the focus figuring out what is fit in that environment. Making the game more tactical as a whole as a result of good level design would have a large benefit on the game I believe. Unfortunately I did not feel I had the knowledge of level design to really be able to do that on my own to the extent that I wanted, and may ask for help if I ever want to improve that.

7.4 What could be added

Several things could be added other than just touch mechanics and additional levels. One thing, which was mentioned in the playtest section, is the ability for players to map their own keys layout. Another thing that I would like to add is addition secondary and second mutations. I think having more options like that could make the game very interesting. Also if the game was ever to be revamped in a large way additional actions for the player to do can be added and these can be effected by new stats leading to the possibility of even more abilities with which to work.

8. Works Cited

"BoxCar 2D." *BoxCar2D*. N.p., n.d. Web. 2014. <http://boxcar2d.com/>.

Chernova, Sonia. "Local Search and Continuous Search." *Intro to Artificial Intelligence* (CS 4341). WPI,

    2014. Lecture.

Chernova, Sonia. "Local Search and Continuous Search." 2014. Microsoft PowerPoint file.

Evolving Artificial Intelligence Lab at the University of Wyoming. "Evolving Soft Robots with

    Multiple Materials (Muscle, Bone, etc.)". Online video clip. YouTube, Apr. 4, 2013. Web. 2014.

Moriarty, Brian. "Perlenspiel | Home." *Perlenspiel | Home*. N.p., 2009. Web. 2014.

    <http://users.wpi.edu/~bmoriarty/ps/index.html>.

9. Level Designs
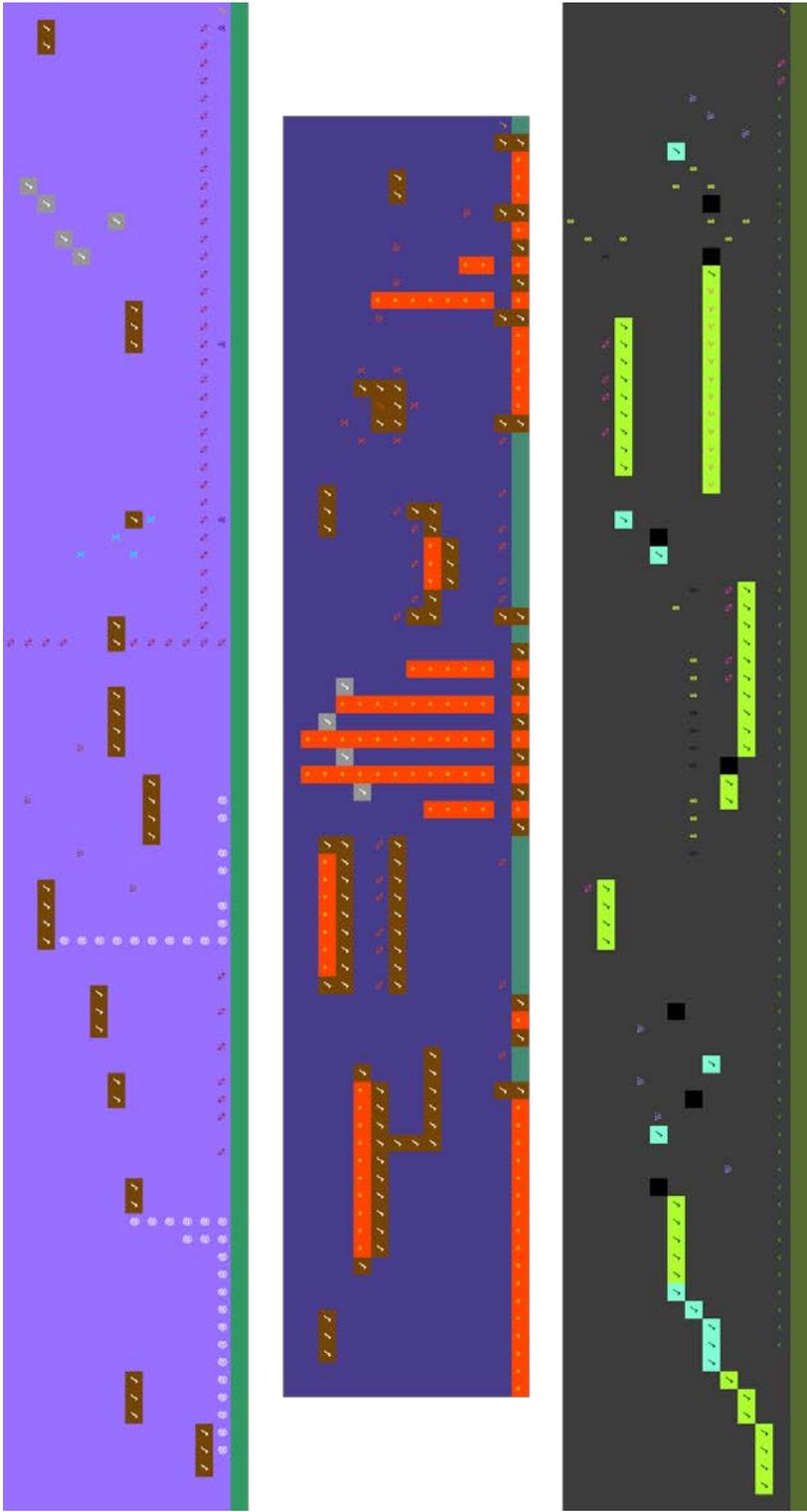
Layout for levels one through three