CMP105 Coursework Report: *Definitely Not Mega Man*

Justin Syfrig

1904770

Git Username: nefastes

Introduction

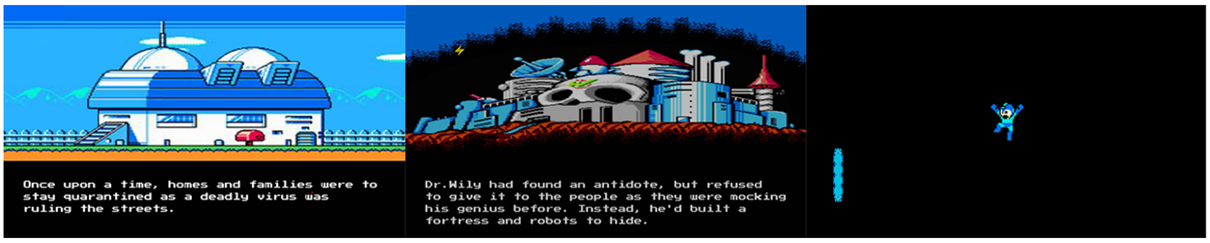
I thought of this project after launching some games on my NES, particularly Mega Man 1, my personal favourite. At first I wanted to make a Cuphead style boss fight, but quickly changed my mind and settled on to recreate similar mechanics and enemies to the original Mega Man game. *Definitely Not Mega Man* is a 2D platformer, in which you play Mega Man, a humanoid that need to defeat the evil Dr.Wily. The game aims at having six levels and one final boss to defeat. The current prototype contains one level and a tutorial level. Each level is divided in smaller sections, and the user progresses through those sections by climbing a ladder or opening doors. *Definitely Not Mega Man* is made using C++ and SFML (Simple and Fast Multimedia Library). SFML is very convenient to handle window events, such as the user inputs, and allows us to build on top of it to handle those events properly. In this way, it is easy to build logics to interact with the user, such as changing menus with correct button presses or mouse placements. SFML gives us a support to play audio sounds and musics, and also provides us with graphical libraries, which are very handy to display in game sprites, like the player, some items or enemies. However, animations, collision detections and responses, and other important elements are not supported by SFML. In this case, I used the provided frameworks by the CMP105 tutors and adjusted them where necessary. *Definitely Not Mega Man*'s major mechanics are ladder climbing, screen transitions and tile detections.

Controls

Mega Man can be moved left or right by pressing **A** or **D** accordingly. Hold **SPACE** to make Mega Man jump. You can do short jumps by briefly pressing it or higher jumps by holding the key for a certain amount of time. To make Mega Man use his blaster and shoot bullets, press either **ENTER** or **LMB** (left mouse button). Note that only three bullets can be fired at a time, like in the original game. Hold **W** to climb up of a ladder or **S** to climb it down. At any point on the ladder, the jump key can be pressed to let Mega Man go off of it. The game can be paused with either **TAB** or **ESC**. To navigate through menus, use the **ARROW KEYS** or the **mouse cursor** to change the selection highlight and hit **ENTER** or **LMB** to make the selection.

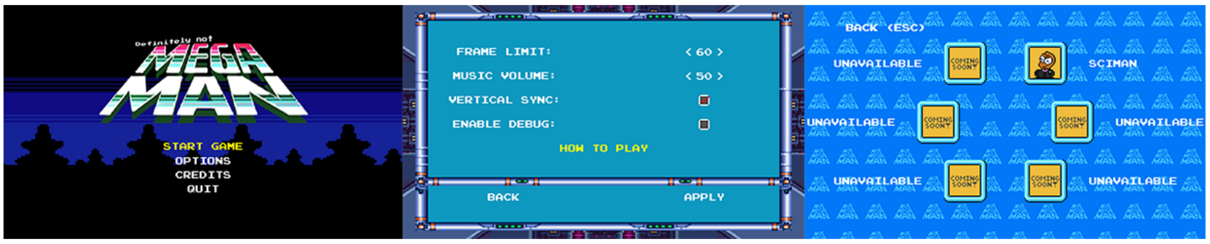
Game Screens

The current prototype contains 7 game screens, which are: the introduction cinematic, the fake credits, the main menu, the option menu, the stage selection menu, the tutorial and the available level. The introduction cinematic and the fake credits can be skipped by pressing either **ENTER** or **LMB**. The option screen allows you to select your desired frame rate limit, adjust the volume, enable or disable the vertical synchronization and debug mode. A collection of screenshots is available below. All non original sprites and images used are referenced at the end of this document.


Once upon a time, homes and families were to stay quarantined as a deadly virus was ruling the streets.

Dr.Wily had found an antidote, but refused to give it to the people as they were mocking his genius before. Instead, he'd built a fortress and robots to hide.

Figure 1: The introduction of the story of the game during the cinematic (left), Dr.Wily's Castle (middle), the end of the cinematic (right)



FRAME LIMIT: < 60 >
MUSIC VOLUME: < 50 >
VERTICAL SYNC: ☐
ENABLE DEBUG: ☐

Figure 2: main menu (left), option menu (middle), stage selection menu (right)

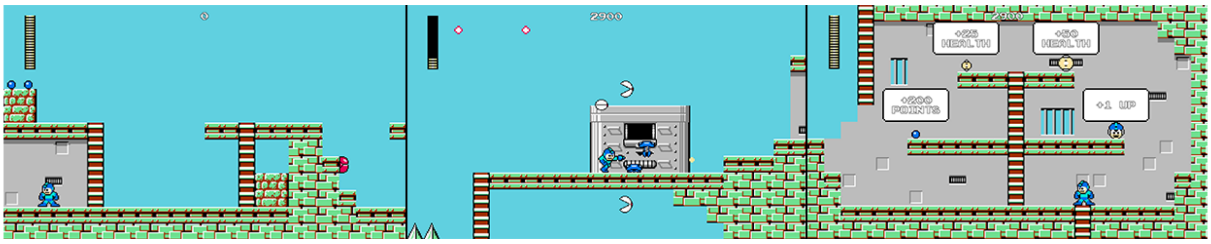


Figure 3: spawn area of the available level (left), Mega Man fighting enemies in the level (middle), a section of the tutorial level showcasing the items of the game (right)

Input

The game handles inputs via an object made by the input framework provided by the module tutors. SFML tells us which keys have been pressed or released between each frame. The input object registers those inputs into an array of boolean (values that can be either *true* or *false*; here *true* if it was pressed, *false* otherwise). We can then identify anywhere in the program if a key has been pressed by referencing the input object where we need it and by calling the following function:

input→isKeyDown(desired key)

However, this could cause issues, like if the user holds a key or while navigating through menus. For instance, if the user would like to change the highlighted selection to the next one and press the corresponding key, the highlight will not be in the intended place. In fact, since the check happens on every frame and the game runs at least thirty frames per second, the user would have to press the key for no more than a thirtieth of a second to move the highlight to the next one. To fix this, I have modified the framework to also contain a second array, which will register all the inputs of the previous frame. To do this we need to call my following function in the main program:

```
input.updatePreviousFrameKeys()
```

It is very important to do this before we start to check key events with SFML, at the very beginning of the next frame calculation, as the normal array of keys has not been modified yet. I also introduced the following function:

```
input→isKeyDownOnce( desired key )
```

When it is called, it will check if the *desired key* was also pressed in the frame before via the new array, in which case it will return *false*, as we would like to have it pressed only once. If the key was not pressed before, it will return *true*, allowing us to change the selection highlight to the next one as intended. This is also used in the game, to navigate through the pause menu, make Mega Man shoot or jump. With this Mega Man shoots only 1 bullet at a time instead of firing them all on each frame and will not repeatedly jump if the user holds the jump key.

Sprite Work

Radically, a sprite is an image that we can manipulate in the game, like the player character or an enemy. To animate my sprites I used the animation framework provided by the module tutors. In short, it is an electronic version of the *Cel Animation* technique. This works by capturing a certain area of a sprite sheet, an image that contains multiple sections of a sprite, after a certain amount of time has elapsed. This allows us to have only one image loaded in memory, and to work with this particular image only instead of having hundreds of them.

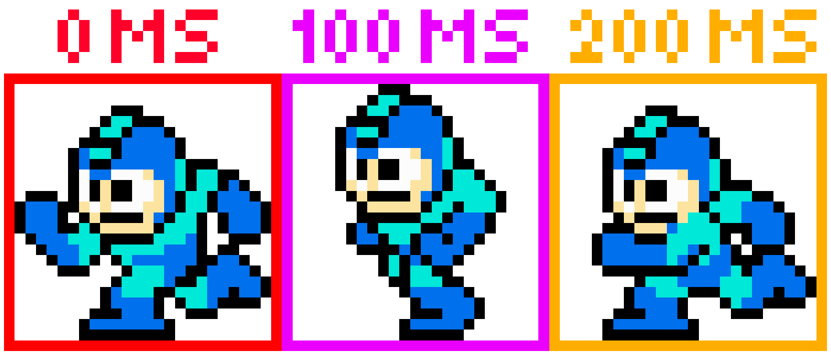
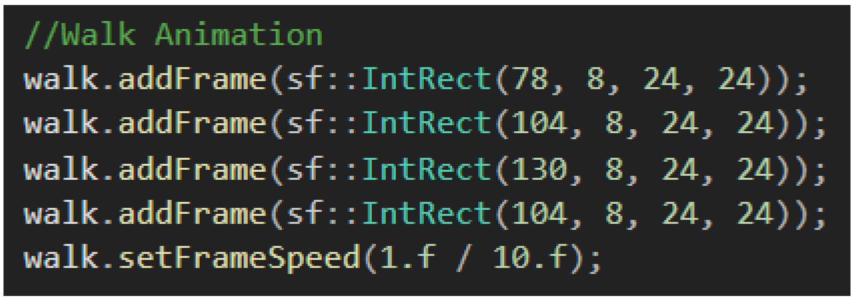


Figure 4: This example shows how the game changes its capture of a certain area of the same sprite sheet after a number of time

In that way, it is possible to make the illusion that the character is moving, climbing, etc. Every animated sprite in *Definitely Not Mega Man* uses this technique. The only difference being that Mega Man animations are controlled by user inputs, such as shooting a bullet or climbing a ladder, and other animations, like the enemies, are determined by the computer. To implement this into code, we need to load both a texture and animation objects. The animation object will only take care of returning the needed rectangle of capture after the desired amount of time has passed.



```
//Walk Animation
walk.addFrame(sf::IntRect(78, 8, 24, 24));
walk.addFrame(sf::IntRect(104, 8, 24, 24));
walk.addFrame(sf::IntRect(130, 8, 24, 24));
walk.addFrame(sf::IntRect(104, 8, 24, 24));
walk.setFrameSpeed(1.f / 10.f);
```

Figure 5: This code shows how I have initialised the walk animation of Mega Man, by first declaring the sequence of rectangles to capture in the sprite sheet and finishing by setting the desired frame speed (here it will change every 100 ms)

The texture object (its position in memory) will be referenced to the player object. Then SFML provides the following function:

setTextureRect(`desired rectangle we get from the animation object )`

It allows to change the rectangle of capture on the referenced texture. If it is not changed, the default capture is the texture size. And this is how an animation appears in game, by constantly changing the area of capture on a sprite sheet.

Collision Detection and Response

The collision detections in *Definitely Not Mega Man* follow the *Axis Aligned Bounding Boxes (AABB)* collision principle. This method will actually check if two rectangle shaped objects are <u>not</u> colliding, as it turns out to be easier to do so. This works by checking every opposite sides of each object, accordingly right-left, left-right, top-bottom and bottom-top. For instance, with the right-left detection, if the right side's position of the first object is smaller than the left side's position of the second object, then they must not be colliding, terminating the detection by returning *false*. To use the AABB method, the tutors provided us with a collision framework, which is using static functions. These static functions can be used anywhere in the program without the need of an object, as they are defined in compiling time rather that during the execution. A simple include of the framework and the following function call will be sufficient:

`if( Collision::checkBoundingBox( object1 , object 2 ) )`

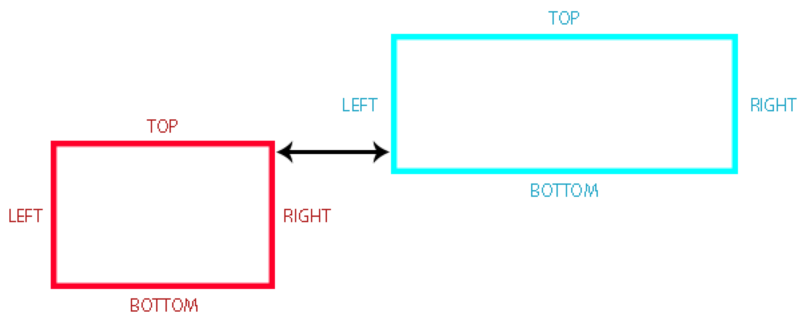


Figure 6: This figure shows two non colliding objects, and how the distance of two sides is being compared

If a collision is detected, hence if our function call returns *true*, we can call a specific collision response. For instance, when I check for collisions between the player and an enemy, and our detection method tells us that the collision happened, we make a collision response to damage the player of a certain amount. Other cases would be the player or enemies colliding with the ground, bullets hitting enemies or colliding with a door entrance. Each response is specific to which objects have collided together. If the player collides with the ground, we want to adjust his position to stay on top of it, and not damage the player like we would do with an enemy.

Audio

SFML provides an audio support. However, it is very difficult to manage multiple sounds and musics at a time. This is why the module tutors made an audio manager framework, which aims to help us declare multiple sounds and musics only once in the entire program. It is then very easy to access those sounds or musics via the manager object.

audio→addMusic(music path, music name)

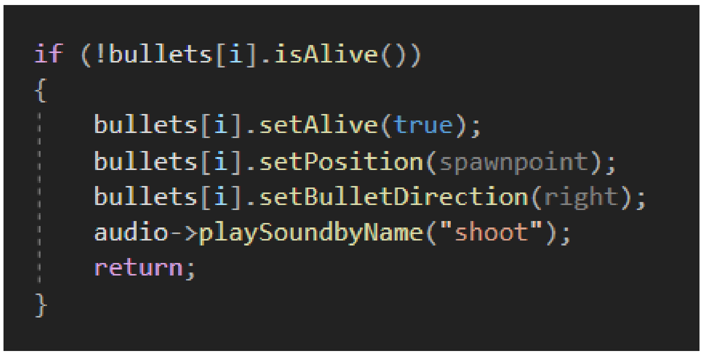
audio→addSound(sound path, sound name)

audio→playMusicByName(music name)

audio→playSoundByName(sound name)

Figure 7: This code example show how sounds and musics are declared, and how they are accessed later on in the program

Unlike sounds, musics will not be loaded into memory but rather streamed from an existing file, as they are considered as large files. SFML will create a new thread for it, which separates it from our program to be executed alone without slowing performances down. This is why only one music can be played at a time, while multiple sounds can be played at the same time. By referencing access to the audio manager object, we can play sounds and musics anywhere in the program. For instance, I have referenced it to my player object, thus I can play a sound when the player has collided with the ground or when he fires a bullet very easily.



```
if (!bullets[i].isAlive())
{
    bullets[i].setAlive(true);
    bullets[i].setPosition(spawnpoint);
    bullets[i].setBulletDirection(right);
    audio->playSoundbyName("shoot");
    return;
}
```

Figure 8: this piece of code shows how the audio is played each time the user fires a bullet

Game Logic and Unique Mechanics

Of all the mechanics of the game, the major one is definitely my tile detection logic. In fact, each section of a level contains small squared tiles, each assigned with a different texture, such as a sky or ground texture. In addition to that, I also gave them identification names like “sky” or “worldSolid”. It helps me to identify what kind of tile the player is colliding with. Each frame, the player will detect which tile is on top, on the right, on the left, on the bottom and in the middle of his position. This technique is used to make the player climb ladders, transition to the next screen while climbing a ladder or falling out of bounds and the door system, no matter their placement on the map.

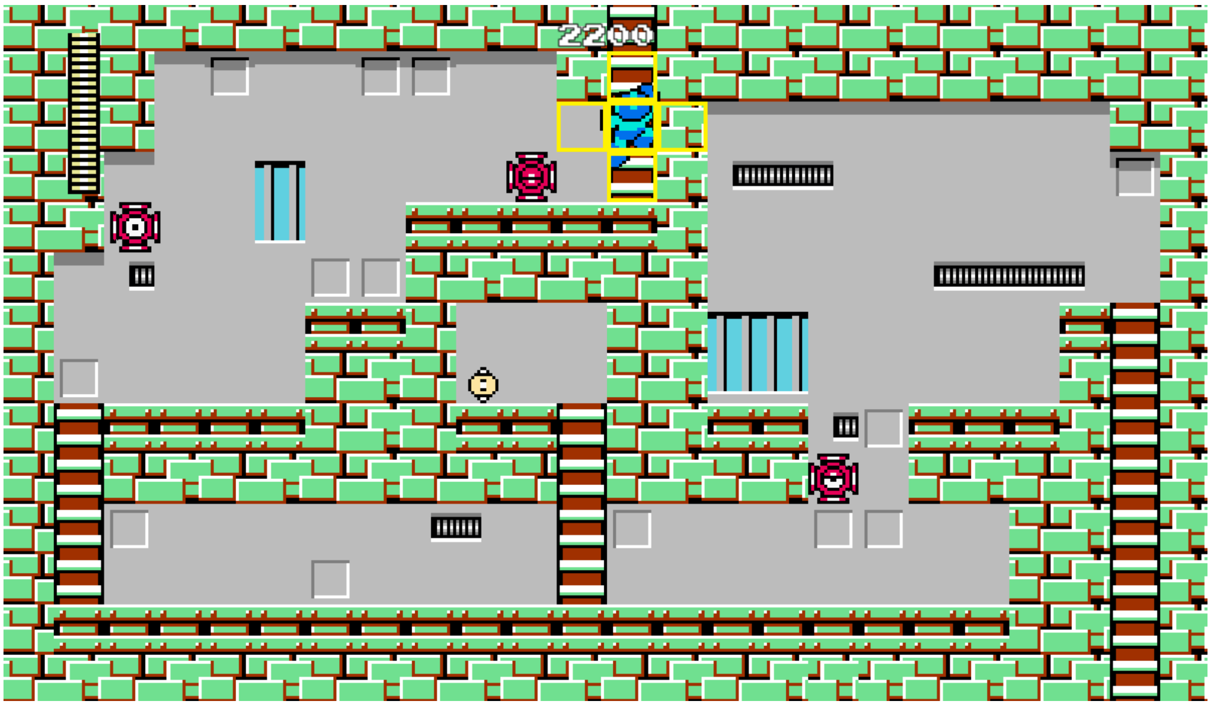


Figure 9: shows an example of the tile detection (in yellow) while climbing a ladder

Making the ladder mechanic is without a doubt what took me the most time to figure out properly, and resulted with this tile detection system. The trickiest part was the mantle animation when the player reaches the end of a ladder, like in the original Mega Man game. The result of my research is the following logic:

- If we are on the ground and more than ¾ of our character collides with a ladder tile, we make the ladder mechanic to be available and ready to climb.
- If the user gives us input to climb a ladder and it is available, start climbing or descending the ladder and also make sure the animation is now the climbing one.
- If we are climbing and the middle tile is not a ladder tile: then we are on the last tile of the ladder and must change the player animation to show a mantle.
- If we keep climbing, the middle tile is not a ladder tile, we are mantling and more than ¾ of our character is above the last ladder tile: then we finish the climb by putting the player on top of the tile and by making sure the program does not consider him to be on a ladder any more.
- If we are climbing a ladder and the middle tile is “none”, it means we are outside of the screen and we should load the next section, start a screen transition, and when it is finished unload the previous section. Only one section is loaded at a time! The same logic applies for

climbing downwards or falling off screen, but in this last case it would be detected with a disguised special tile, looking like a sky tile.

One last remark would be that normally ladder tiles are non colliding. However, in the original game, it is possible to stand on the top tile of a ladder. If I made all ladder tiles solid, the player could stand on any tile, making the ladder completely obsolete. So instead, I took advantage of my tile detection logic and implemented the following:

- If we are colliding with a ladder tile, the bottom tile is a ladder tile but all left, top, right and middle tiles are not, then it means we need to have a response to put the player on top of the ladder tile to make him stand on it.

Conclusion

Looking back at it from now, those ladders were such a gratifying step after being confused with them for weeks. I mean, my Github repository shows that I have “fixed them for real” multiple times, because they never were. I am glad I ended up figuring this out. I still have a two unsolved mysteries though. For some unknown reason moving the screen to a half position (for instance 102.5, 100.5) will make dead black lines appearing on the screen. The other was that apparently, if the program contains too many layers (for instance an object inside an object inside an object etc.) will cause the texture pointers to be corrupted. I could not properly solve those two issues, but I made sure they were avoided. If there are two things I have learned making this project, is that making a game takes a lot of time, and that doing it without using object orientation programming and especially inheritance of classes is unimaginable. I will definitely try to improve my tile collisions in the future, since as of now some tiles will be checked even though the player will never be able to reach them. I will actually try to push this further, as I wonder if it would be possible to identify which sides of a tile can be accessed and collide with the player and which sides cannot. That would definitely improve the game’s performance. As it was my very first experience of developing a game, I had much trouble in the first half of development, but I am looking forward to keep practising my programming skills to make awesome games in the future.

References

Textures & Images:

Note: Some textures and images may have been modified or adjusted through Adobe Photoshop.

StarSimsUniverse (2012) *Dr.Light (Mega Mania Style Game Boy Advanced Sprite)*. Available at: http://forum.rockmanpm.com/index.php?topic=6849.0 (Accessed: 3 April 2020).

StarSimsUniverse (2012) *Dr.Wily (Mega Mania Style Game Boy Advanced Sprite)*. Available at: http://forum.rockmanpm.com/index.php?topic=6849.0 (Accessed: 3 April 2020).

Xtremertro (2018) *Infogrames Entertainment*. Available at: https://xtremeretro.com/infogrames-entertainment-parte-4/ (Accessed: 3 April 2020).

TheAlmightyGuru (2010) *Mega Man 2 - NES - Weapons*. Available at: http://www.vgmpf.com/Wiki/index.php?title=File:Mega_Man_2_-_NES_-_Weapons.png (Accessed: 24 April 2020).

Mystick10 (2012) *Mega Man - Shiver Man's Stage*. Available at: https://www.deviantart.com/mystick10/art/Mega-Man-Shiver-Man-s-Stage-IN-PROGRESS-297333159 (Accessed: 31 March 2020).

Unknown Artist (Unknown date) *Megaman Logo*. Available at: https://logot.org/megaman-logo/megaman-logo-39570/ (Accessed: 31 March 2020).

Nokii (2019) *Wily Castle in Mega Man 2*. Available at: https://www.ssbwiki.com/File:Wily_Castle_MM2.png (Accessed: 3 April 2020).

Zmanwarrior (2012) *Dr.Light's House (extended)*. Available at: https://www.smackjeeves.com/discover/detail?titleNo=91833&articleNo=381 (Accessed: 3 April 2020).

hfbn2 (2010) *Dr.Light Lab Inside*. Available at: https://www.deviantart.com/hfbn2/art/Dr-Light-Lab-Inside-182996044 (Accessed: 3 April 2020).

Evan G. (2016) *Megaman X3 Pause Menu*. Available at: http://snescentral.com/review.php?id=0054&num=1&fancy=yes&article=proto (Accessed: 31 March 2020).

DanFo07 (2015) *Megaman 3 Stage Select*. Available at: https://danfasulo.wordpress.com/category/mega-man-land/ (Accessed: 4 April 2020).

Random Talking Bush (2011) *Mega Man and Friendly NPCs*. Available at: https://www.spriters-resource.com/nes/mm/sheet/36582/ (Accessed: 2 April 2020).

Polar Koala, Shadowman44 (2010) *Tiles*. Available at: https://www.spriters-resource.com/nes/mm/sheet/260/ (Accessed: 2 April 2020).

Elnock200, Superjustinbros (2019) *Life and Energy Bars*. Available at: https://www.spriters-resource.com/nes/mm/sheet/113678/ (Accessed: 2 April 2020).

Polar Koala (2009) *Intro & Stage Select*. Available at: https://www.spriters-resource.com/nes/mm/sheet/258/ (Accessed: 2 April 2020).

-ShyGuy- (2013) *Enemies*. Available at: https://www.spriters-resource.com/nes/mm/sheet/32924/ (Accessed: 2 April 2020).

Magma MK-II (2012) *Items*. Available at: https://www.spriters-resource.com/nes/mm/sheet/45633/ (Accessed: 2 April 2020).

AlphaSystem (2010) *Portal Wallpaper*. Available at: https://wall.alphacoders.com/big.php?i=31312 (Accessed: 10 May 2020).

TechOnTheNet (Unknown date) *Letter W key*. Available at: https://www.techonthenet.com/clipart/keyboard/letter_w.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) *Letter A key*. Available at: https://www.techonthenet.com/clipart/keyboard/letter_a.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) *Letter S key*. Available at: https://www.techonthenet.com/clipart/keyboard/letter_s.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) *Letter D key*. Available at: https://www.techonthenet.com/clipart/keyboard/letter_d.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) *Space bar key (small)*. Available at: https://www.techonthenet.com/clipart/keyboard/space_key_s.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) Enter *key*. Available at: https://www.techonthenet.com/clipart/keyboard/enter_key.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) Esc *key*. Available at: https://www.techonthenet.com/clipart/keyboard/esc_key.php (Accessed: 25 April 2020).

TechOnTheNet (Unknown date) *Tab key (version 2)*. Available at: https://www.techonthenet.com/clipart/keyboard/tab_key2.php (Accessed: 25 April 2020).

FreePik (Unknown date) *Mouse Left Button Free Icon*. Available at: https://www.flaticon.com/free-icon/mouse-left-button_32041 (Accessed: 25 April 2020).

Audio:

Note: Some audio files may have been modified or adjusted through Audacity.

Joueur Du Grenier (2013) *Chanson Infogrames*. In: *Les jeux Disney [time code: 17:58]*. Available at: https://www.youtube.com/watch?v=5I7pukuy8sQ (Accessed: 3 April 2020).

Gbelair (2009) *Mega Man (NES) Music - Elec Man Stage*. Available at: https://www.youtube.com/watch?v=CJdFxTOysjo&list=PL7EF_qp0zBDmKNoUhxqH7qwDOg_mcmLR4&index=2 (Accessed: 26 April 2020).

Gbelair (2009) *Mega Man (NES) Music - Cut Man Stage*. Available at: https://www.youtube.com/watch?v=GsXEqfDL40k&list=PL7EF_qp0zBDmKNoUhxqH7qwDOg_mcmLR4&index=3 (Accessed: 9 April 2020).

Wireboom (2010) *Mega Man 2 Intro Opening Theme HQ*. Available at: https://www.youtube.com/watch?v=ZT9DST_M_g8&list=PL7EF_qp0zBDmKNoUhxqH7qwDOg_mcmLR4&index=12 (Accessed: 3 April 2020).

SuperChaosControl (2015) *Megaman 1 - Sound Effects*. Available at: https://www.youtube.com/watch?v=nVpjIcw1spw (Accessed: 9 April 2020).

Cepblu2206 (2010) *Mega Man 1 NES Music: Level/Boss Selection*. Available at: https://www.youtube.com/watch?v=dbKCDrovBNQ (Accessed: 4 April 2020).

Crowman85 (2010) *Megaman Stage Start Collection*. Available at: https://www.youtube.com/watch?v=RKgDZrTAVrM (Accessed: 4 April 2020).

SuperChaosControl (2015) *Megaman 1 - STAGE CLEAR*. Available at: https://www.youtube.com/watch?v=YGjR8iT7-lo (Accessed: 26 April 2020).

Code:

John Dibling (2009) *Random Float Number Generation* [C++ code]. Available at: https://stackoverflow.com/questions/686353/random-float-number-generation (Accessed: 6 April 2020).