CS4067 – Writing Games Analysis

C++ Project

Karma

by: Gearoid Sugrue - 11139102

Table of Contents

Introduction

I decided to develop a game in C++ in order to improve my skills. Despite previously doing a module in C++ I never grasped many important features about C++. I feel that is important to spend more time programming in C++ as it is the video-game industry standard and is one of the first things many software companies look for in potential employees, even if they don't programme in C++. I also decided to try and link the project back to the CS4067 module by thinking outside the box and incorporating a theme as opposed creating something generic and unoriginal.

How to Measure My Improvement

While the code itself serves as good indicator of what level my C++ skills are I decided to also track my improvement via other means. To help measure my improvement I answered a series of short questions based on different aspects of C++ both before I started on the project and after I completed it. The questions were taken from the website IndiaBix[a] and were broken into five different sections, Object-Orientated Programming Concepts, Functions, Constructors, References and finally Objects and Classes. I answered ten questions from each section and completed an additional general C++ knowledge test from MyCQuiz.com[b] which comprised of a further thirty questions. The results of of the tests will be discussed later.

Tools

GitHub

GitHub is a code management tool. It is primarily used to allow multiple people work on the same project at once but it is also useful for rolling back the code if the code gets broken and is too time consuming or difficult to repair. Even though it was unnecessary, it was a good opportunity to improve my GitHub skills as I was not familiar with it.

Allegro 5

Allegro 5 is a set of video-game programming libraries for C/C++.[c] It helped me with the graphics side of the game.

Code::Blocks IDE

Code::Blocks is an open source, cross platform IDE that I used to write my code for this project.

Primary Objective and Goals

The primary objective for this project was to improve my C++ skills. The next goal was to create a working prototype of a game. It was for these reasons that I mainly focused on creating systems for a game that could be reused at later in date or easily modified to create an entirely new game as opposed to creating a game with a lot of content. The main purpose of the game-play is to show how each of the systems work and interact.

About The Game

Despite my focus being on the background systems that make a game I decided to incorporate a theme into the game. My decision to do this was influenced by both the nature of the CS4067 module itself and by game designer Brenda Romero, who visited the University of Limerick at the start of the semester. During her presentations she spoke about serious games and how they have an important point to make. While I'm not going to be covering an overly serious subject I decided it would be beneficial and more suited to the module to have a theme in the game.

The game is about karma; how the choices you make effect you and the world around you. The game involves the player going out to complete a simple everyday task. As the player moves along they encounter situations where they have to make choices, with each choice positively or negatively effecting the character's karma. The game focuses on two main aspects of karma.

1. How Karma Effects Probability

The first is how karma effects and the probability of certain events happening. It will be broken into two categories. Events that karma can influence. For example, being nice to a character in the game increases the probability of that character being nice to you. And events that karma can't influence. This message will be conveyed by events that will occur at set points during the game whose outcome won't change regardless of the players karma level. For instance, the player will participate in a lottery and lose every time.

2. How Karma Effects Our Mental State

I believe the choices we make also affect our own personalities and outlook on life. Bad moral choices can lead to a more bitter, angry or unpleasant

personality and vice versa. This aspect was be explored using the player's avatar. The player is represented on screen as a coloured rectangle. The colour of the rectangle represents both the player's karma and mental state. If the player does a helpful action their positive karma increases and the rectangle's hue will change to reflect this, likewise unhelpful actions result in a decrease in karma.

To represent the character's outlook on life the colour of the environment changes slightly. I believe nice people see others nicer light while bitter people tend to see others as also being bitter. The screen turns a few shades duller if you make negative choices. The opposite can also happen, when you make positive choices the screen gets brighter.

The Use of Colour

Colour is used to represent the player's karma level. I've researched the colours that people associate with good and evil and have found that different colours can be mean different things depending on a wide variety of factors. However, I have found that in general people tend to associate blue with being pure and good. [d] For that reason the characters hue will turn blue when positive actions are carried out. Similarly I found that red can be seen as bad, hence negative choices will cause the hue to turn red.[e] Likewise, I have chosen gray as a middle ground between positive and negative as it is often associated with neutrality.[f]

Non-Implemented features

Due to time constraints and the primary focus of the project being on improving C++ some of the features I was considering weren't implemented. These are features that I would like to implement in the future as I believe they would provide additional value to the game.

Firstly, is how other characters view you. In the game, similar to real life, the other characters can only see what's before them. They wouldn't know what your really like. Instead they would make judgments based only the acts that they've seen you commit. To represent this fact I've thought about giving the character a 'skin'. The border of the rectangle will be the colour the NPC character thinks you are. You may appear as a good person to the character your talking to despite having negative karma.

Finally, aside from conveying the message of what I believe karma is, another objective I'd like the game to have is to get the player think about what colour they are. When playing games people often make moral choices that aren't reflective of their true nature. I would like to implement an optional personality test before the game starts. The answers would then be used to calculate player's own karma level and colour and when the game is finished it will be shown alongside their in-game character's colour.

Description of the Code

The *main* class initializes Allegro5 and also contains the game loop. When this loop is finished running the game will close. The loop also listens for certain keyboard events, such as the *ESC* key being released and an instance of the ScreenManager class has its *Update* and *Draw* function called repeatedly. The *Update* function is called whenever something has changed or every sixty seconds, similarly the *Draw* function is called sixty times every second. Both of these are carried out by the use of timers and event listeners.

The *ScreenManager* class is responsible for switching screens and for invoking the screen thats currently active *Draw* and *Update* functions. There are numerous screens, like *SplashScreen* and *TitleScreen*, that inherit from *GameScreen*. *GameScreen* acts like an abstract class as its relies on its subclasses to provide implementations for most of its functions. The reason for this is to allow *Screenmanager* to simply call *Draw* or *Update* on the current screen without having to check what kind of screen it is.

The *Animation* class is responsible drawing objects and characters to the screen. The *FadeAnimation* class is used to smoothly transition between screens by fading in and out. The *SpriteSheetAnimation* class is for drawing sprite based images. Both *FadeAnimation* and *SpriteSheetAnimation* are sub-classes of *Animation*.

The *GameplayScreen* is where the game map is loaded and drawn to the screen. It contains a lot of the game's logic such as controlling the game's camera and *Entity* collision.

The *Player* class contains details of the player's character and also checks for certain key press events. The *Entity* class contains the details for different types of objects, such as characters and signs. The *EntityManager* class is responsible for creating and loading all of the different entities as well as invoking each their various functions like *Draw* and *Update*.

Critique of the Code

While I'm pleased with the how the code turned there are some issues with it. The first being the *SpriteAnimation* class. Too many late changes to the how the class worked resulted in the *SpriteAnimation* class not displaying sprites correctly. Because the issue arose so late I didn't have the time to fix it. This meant I could no longer use this class and instead had to use a quick and messy work around to display any sprites.

The second issue was the way the map was handled. This is currently done in the *GameplayScreen,* however it would be more ideal if the map had its own set of classes dedicated to loading, drawing and updating it. It would have also made

made collision detection easier which is my next issue. Collision detection in the code is quite poor. It is simply hard coded in to stop the player falling off the screen. Ideally collision detection would work off both the map tiles and *Entity* objects. This would mean that if the map or objects were changed the collision detection would need very little, if any, changes.

The way the *Entity* details are loaded from their *txt* files is also not ideal. It would be better if there was a set of classes that would load details from any *.txt* file.

The *Player* class was originally supposed to inherit from the *Entity* class in a similar way that the various screens inherit from *GameScreen* and the current contents of the *Entity* class was to be split into similar sub-classes like *Player*. This would have allowed separate classes for Characters and Objects which would get rid of the messy conditional logic thats currently in the *Entity* class. It would have allowed the sub-classes to override the *Entity* class functions for greater flexibility.

Another change I would like is a better timer system. Currently, there is one main timer. I would have being interested in implementing a system that could effectively use multiple timers to allow for different speeds for features like animation.

Finally, there are numerous areas throughout the code that could be further optimized. Specifically, at the moment there is a lot of conditional logic and even separate functions for dealing with both *OBJECT* and *CHARACTER* type *Entities.* Ideally I wouldn't have to keep checking what type of object each *Entity* is. As I mentioned above having sub-classes of the *Entity class* would help this.

Tests Results and Discussion

1) **Object-Orientated Concepts**

   Before: 4 out of 10.

   After: 6 out of 10.

2) **Functions**

   Before: 4 out of 10.

   After: 7 out of 10.

3) **Constructors**

   Before: 7 out of 10.

   After: 8 out of 10.

4) **References**

   Before: 6 out of 10.

   After: 8 out of 10.

5) **Objects and Classes**

   Before: 7 out of 10.

   After: 7 out of 10.

6) **General C++ Knowledge**

   Before: 11 out of 30.

   After: 23 out of 30.

Both before and after tests lasted approximately two hours. The most noticeable improvement was in my general C++ skills. There was also marked improvement in Functions and References which I used often when working on the project. I did quite well in in some areas in the first test as there their was some overlap between other programming languages such as Java.

Conclusion

My main objective for doing this project was to improve my C++ skills as much as possible. Overall I found developing a game in C++ an effective way to improve my skills as shown by both the complexity of the code and the results of the on line C++ tests that I completed. Creating a game in C++ required the use and understanding of many C++ features. I found that as I was creating the game I was constantly finding better ways to do things and each time I had to push my skills passed their limit. It was often frustrating as C++ can punish you for your mistakes but I also had many moments where I saw just how much my skills had improved. Creating a game also means that I have to something to show for my work, whether it be to potential employers or simply to friends. Overall I'm more confident in my C++ skills as I feel they have improved significantly since the start of this project.

References

[a] – IndiaBix Technologies (2008) *IndiaBix* [online], available: http://www.indiabix.com/cpp-programming/questions-and-answers/ [accessed 1 Dec 2014].

[b] – Sourceforge (2014) *Allegro* [online], available: http://alleg.sourceforge.net/ [accessed 10 Dec 2014].

[c] – My C Quiz (2007) *My C Quiz* [online], available: http://www.mycquiz.com/ [accessed 1 Dec 2014].

[d] – QSX Software Group (2002) *See color theory in action* [online], available: http://www.color-wheel-pro.com/color-meaning.html [accessed 2 Dec 2014].

[e] – TV Tropes Foundation (2012) *Good Colors, Evil Colors* [online], available: http://tvtropes.org/pmwiki/pmwiki.php/Main/GoodColorsEvilColors [accessed 2 Dec 2014].

[f] - empower-yourself-with-color-psychology.com (2009) *The Color Gray* [online], available: http://www.empower-yourself-with-color-psychology.com/color-gray.html [accessed 2 Dec 2014].

Acknowledgements

Parts of the code were taken or based off code from the *CodingMadeEasy Allegro5 Tutorials* on You tube. They can be found here: https://www.youtube.com/channel/UCas000yWtwjvFzD2zB9Nzmw

The *TextDialogBox* code was taken from *StansAllegroBlog*. Which can be found here: allgrostan.wordpress.com

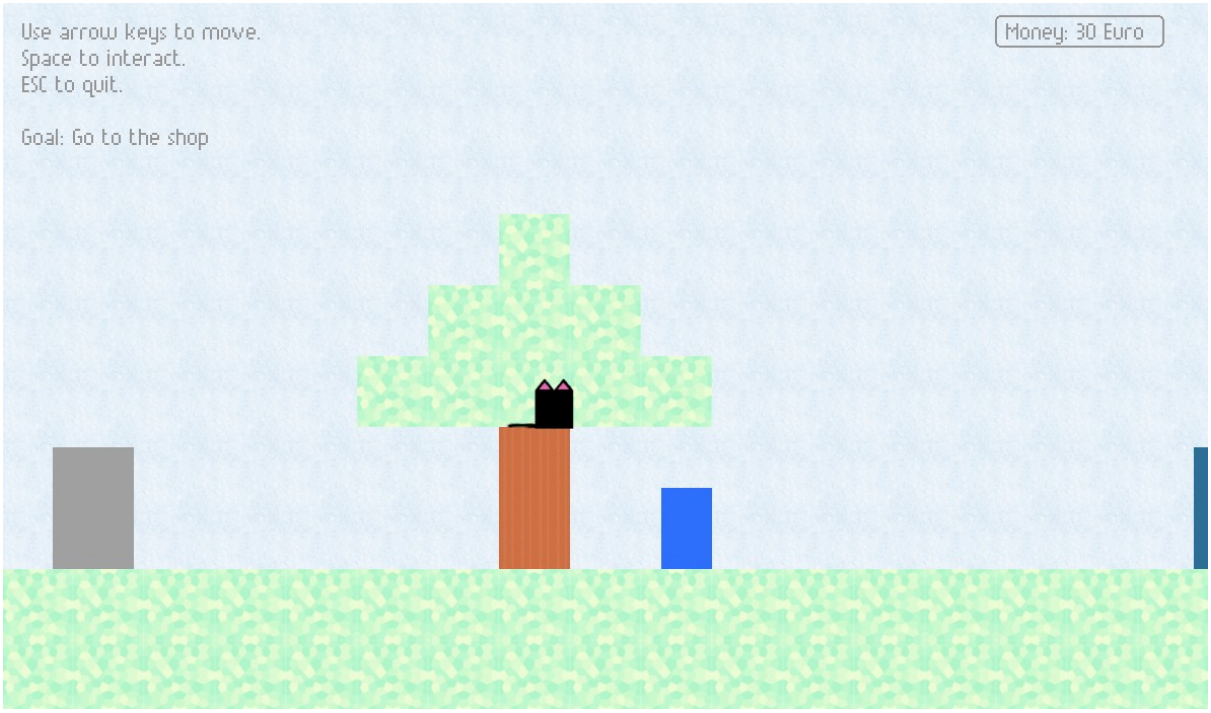
The grass, sky and woof patterns were taken from *subtlespatterns.com.* Which can be found here: *subtlespatterns.com*

Screenshots

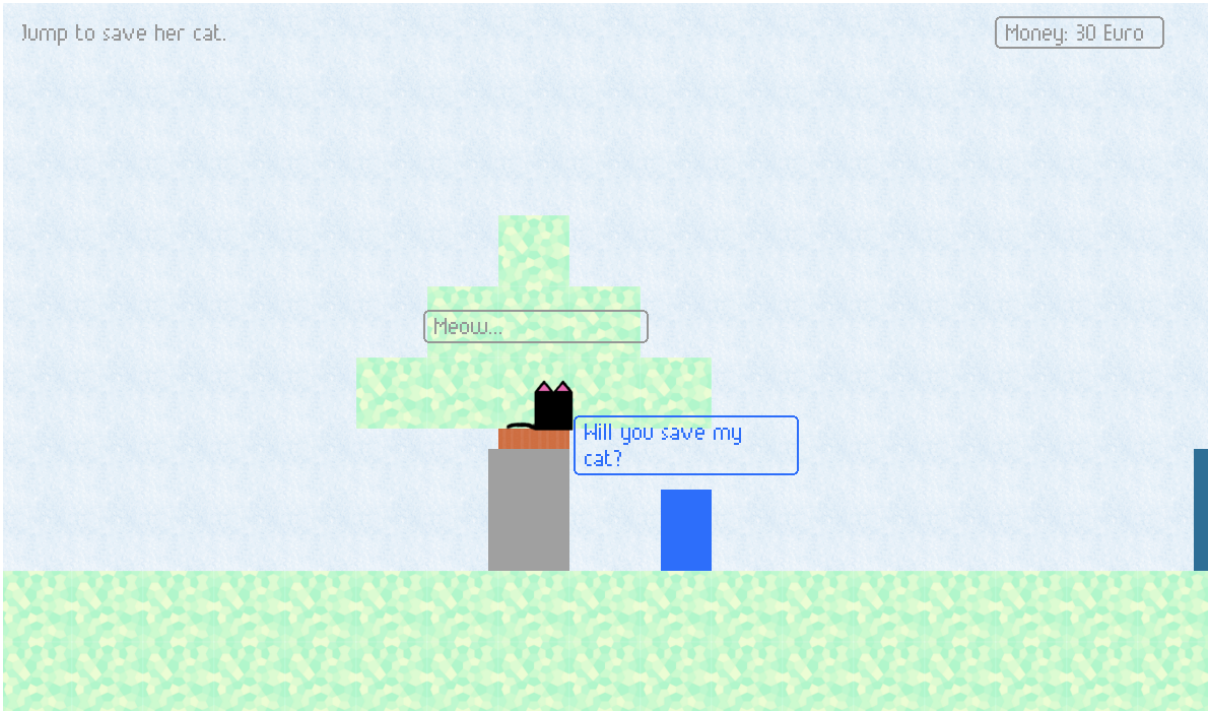
This is the splash screen that is shown when the game is run.

It then continues to the title screen. The logo represents the different levels of karma you can have.

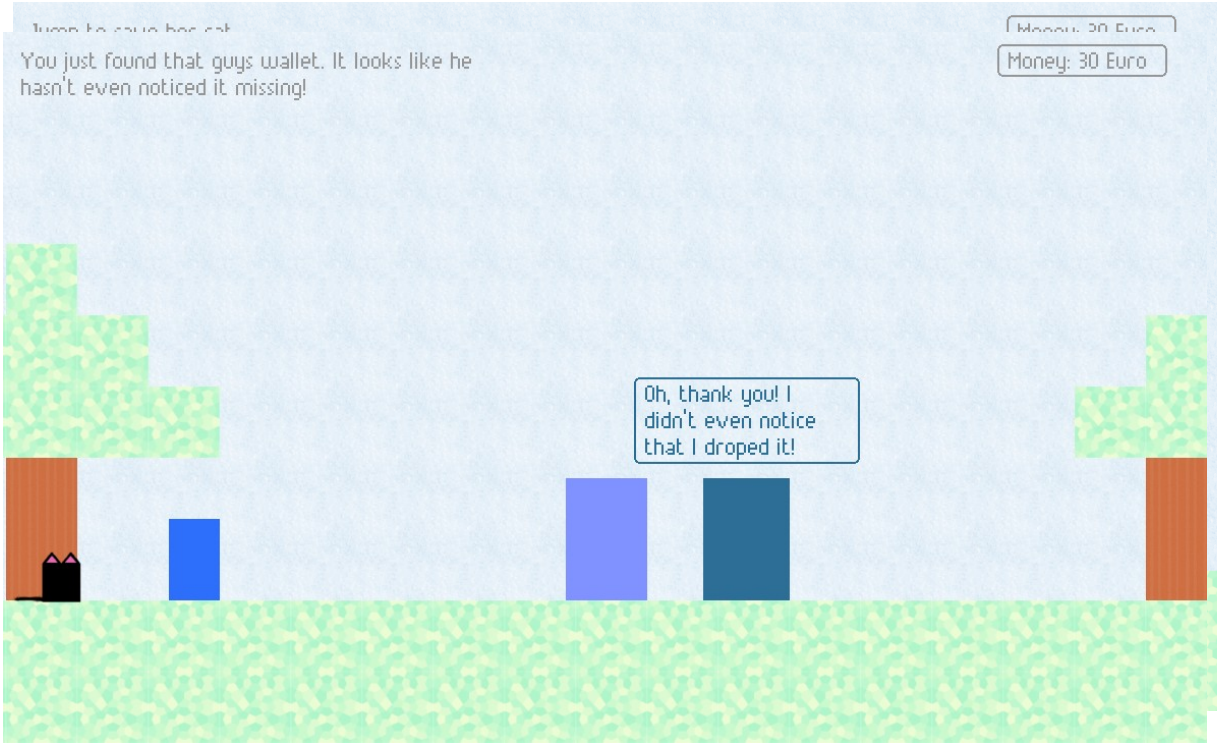
This how the game begins.

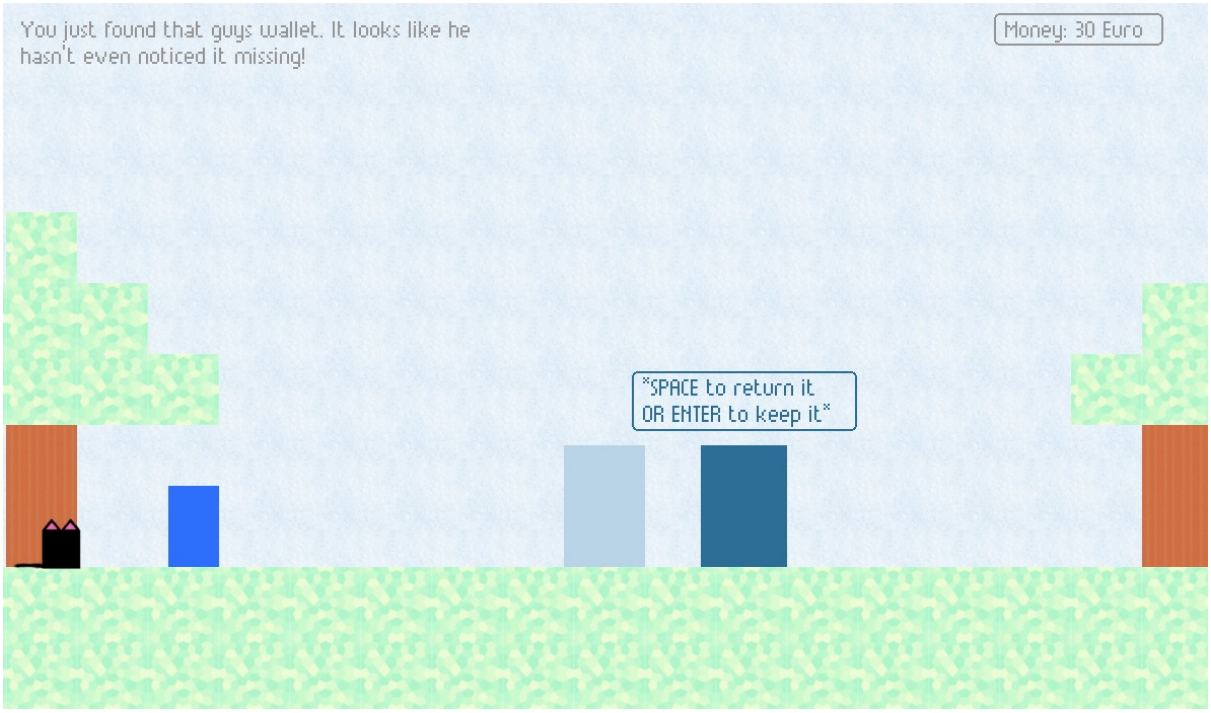


Approaching characters and objects displays their dialog text.

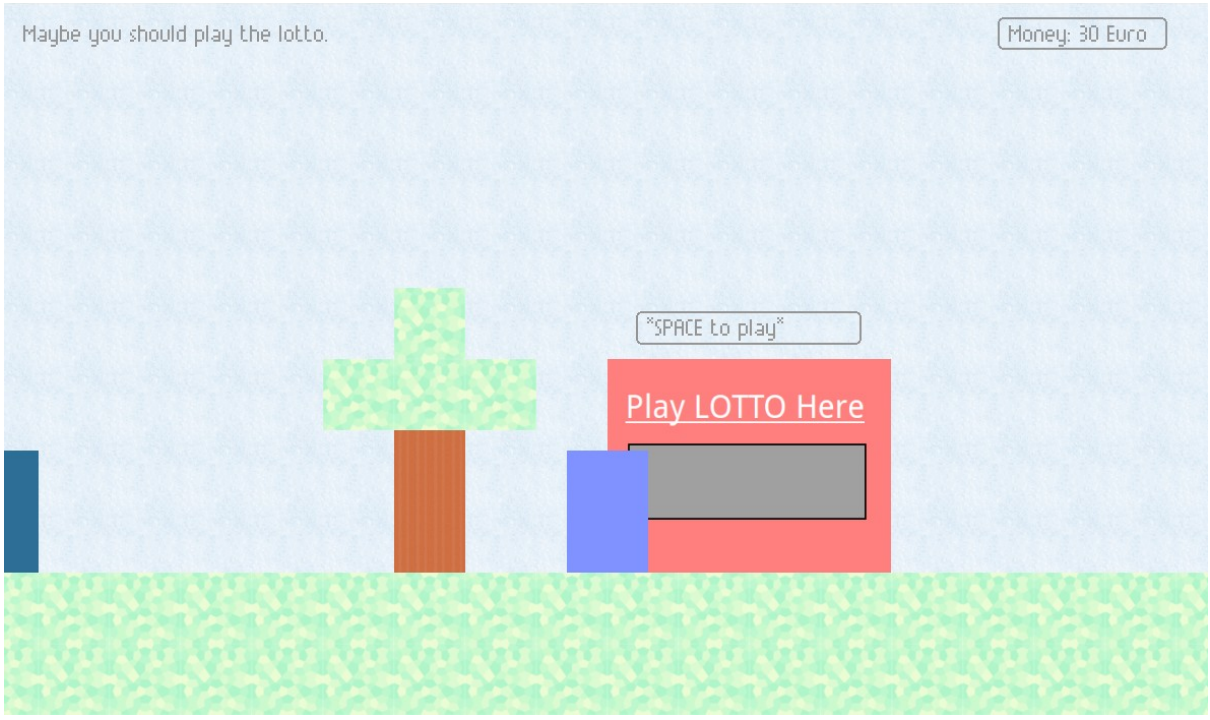


Helping characters increases your karma and your hue turns more blue.

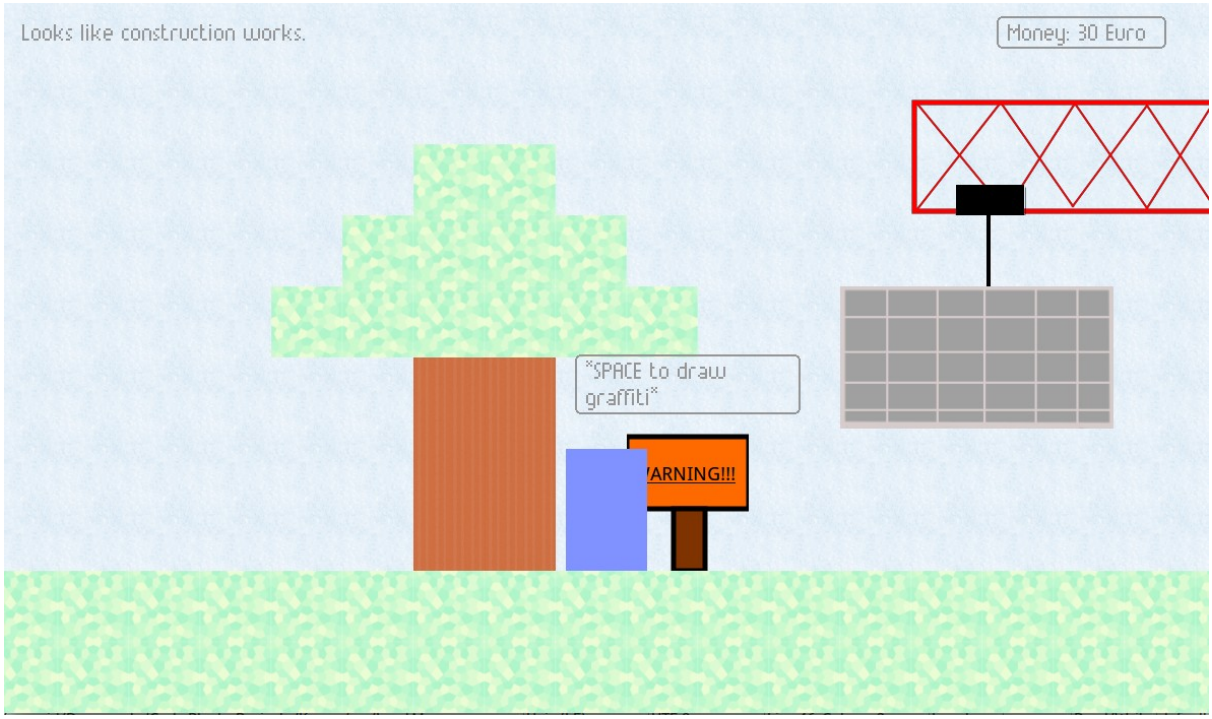




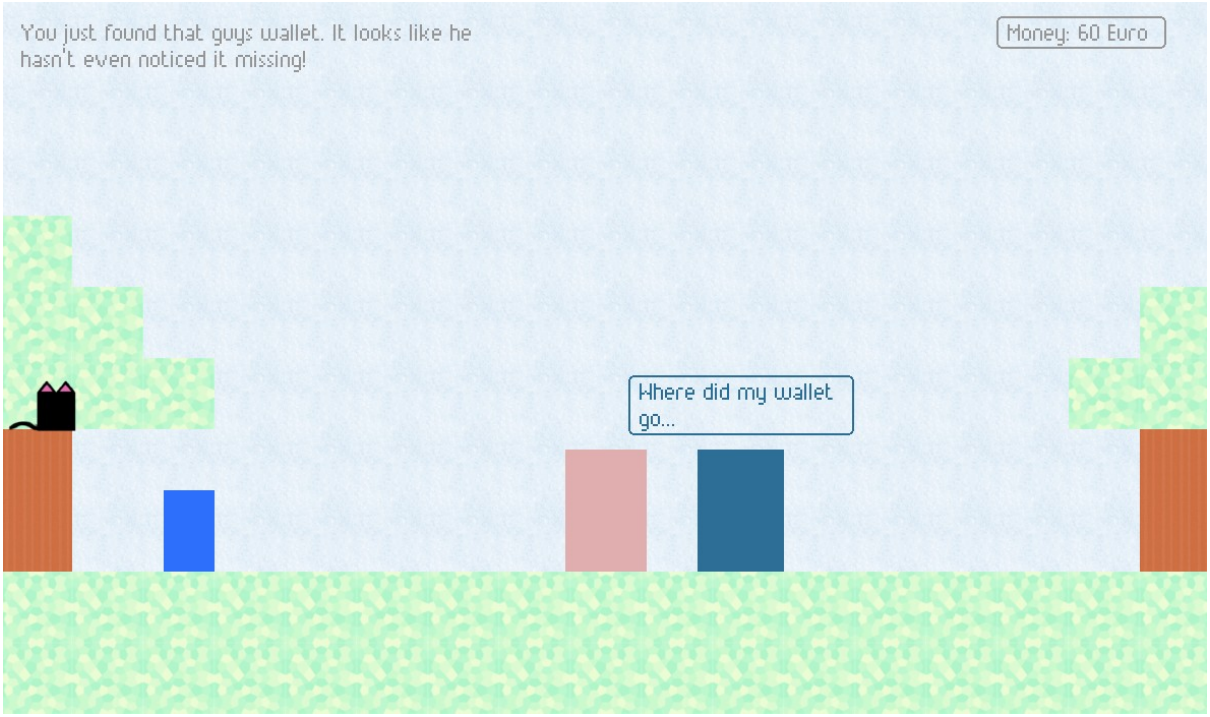
You also can try play the lotto. You lose every time regardless of your karma level. It also costs money to play.

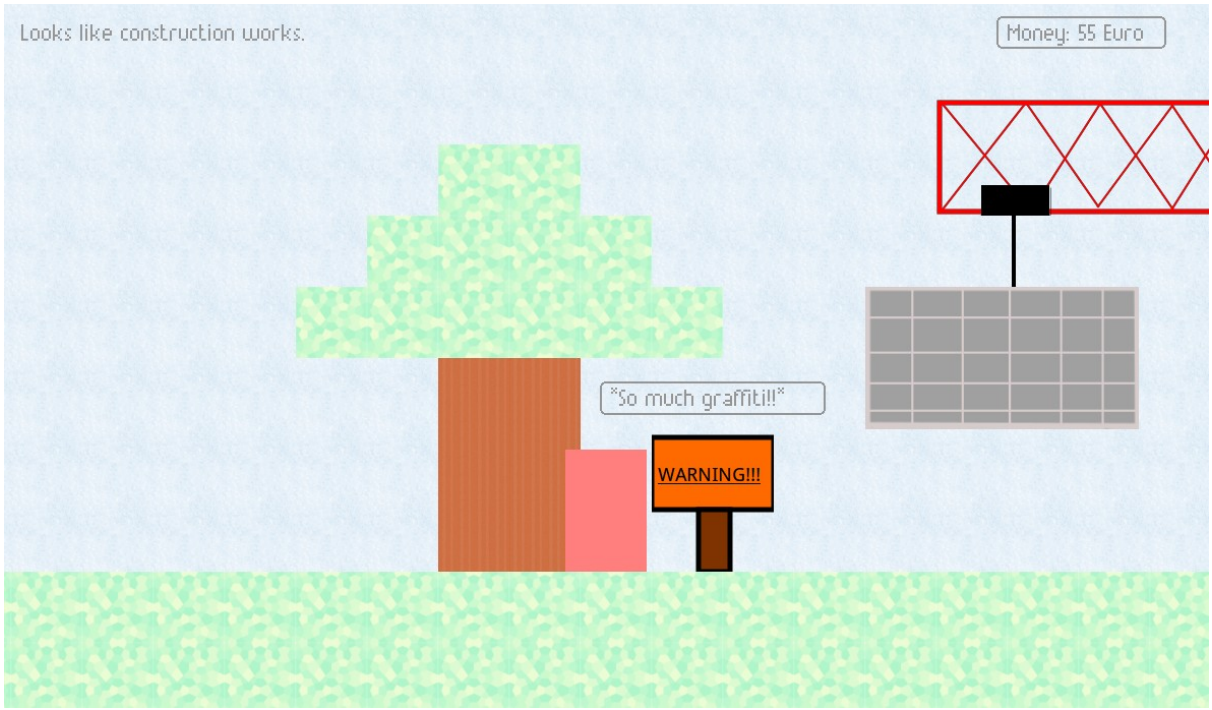


The game ends when you go under the gray rectangle. It represents a ton of blocks thats is being carried by a crane. It falls and kills you and the end screen is displayed.



Below is an example of having bad karma from carrying out negative tasks.





This is the end screen. You can restart the game or quit from here.

GAME OVER. PRESS 'ENTER' TO START AGAIN OR 'ESC' TO QUIT.