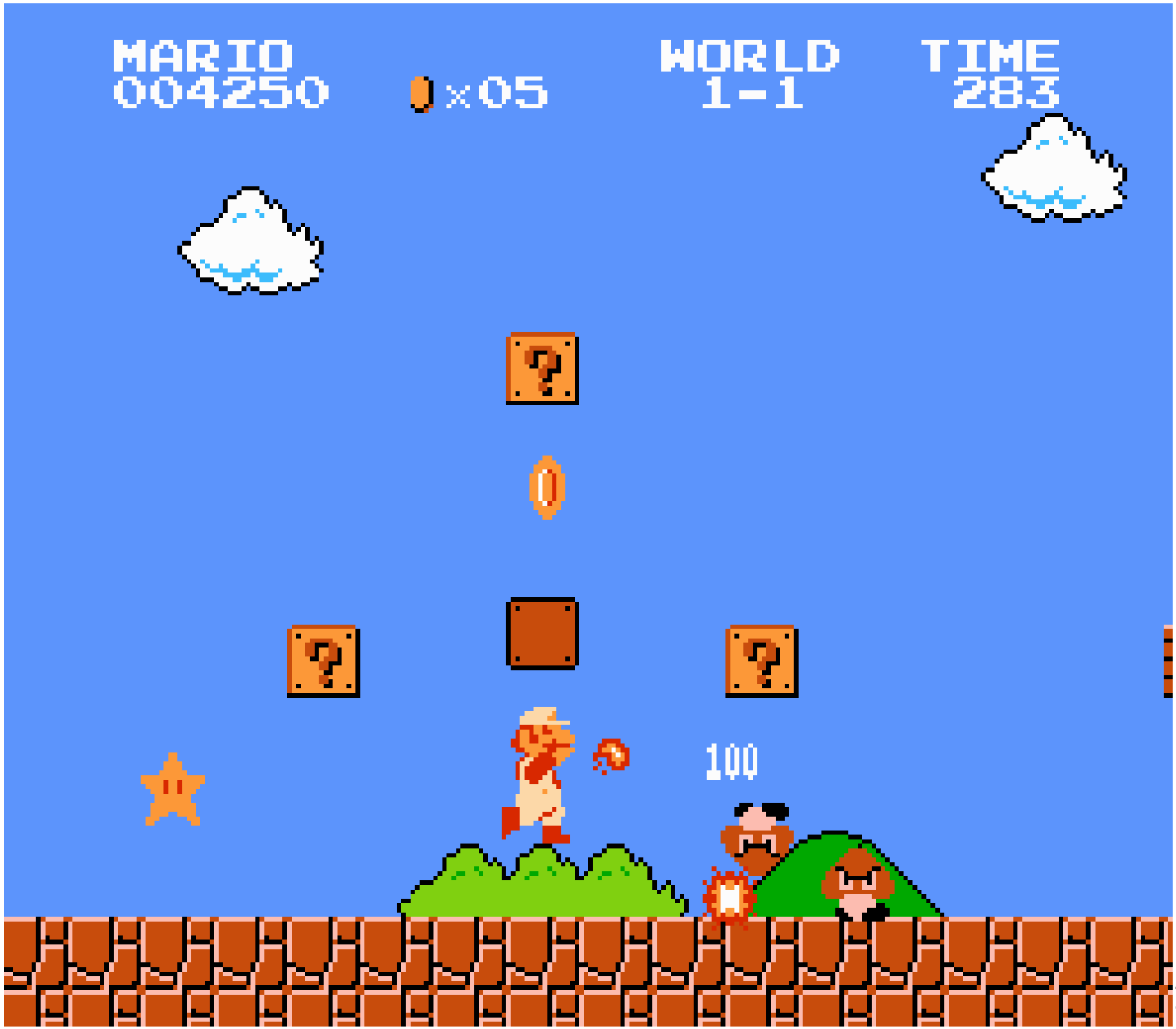
Assessments: Assignment

There is one assignment in three stages. After the first stage, each builds on the work of the previous one.

The assignment is to build the *flying stickman game.*



[1]Super Mario Bros. http://en.wikipedia.org/wiki/Super_Mario_Bros.

Hell is Other People

's code. (misquoted from Sartre.)

The first stage you will do yourself.

At stages two and three, you will extend someone else's code from your class. Your tutor will determine whose. Make sure your code is nice to maintain...

There is no group coding in this course[2]

[2] Yay!

But how will it be graded?

And while we're at it, what if I get some code I really hate, or that doesn't work?

1. We will attempt to find code that works. If we can't then that doesn't bode well.
2. The grade for Stages after the first will be based on what you *add* as well as how well you work with the existing code (simply replacing previous code with your own is not acceptable).
3. The assessment for the first Stage will be easy because we're assessing the difference between *nothing* and your code.
4. Marks for subsequent stages will be based on the *difference* between your code and its previous version.

◇ The Three Stages of the Assignment ◇

Stickman is all alone

Stage one: A lonely stickman

Write a graphics program using Qt Creator to model the basic dynamics of a side-scrolling game

There will be a small text ".config" file in which the details of the stickman's initial position, their size, and the speed they are traveling, along with the dimensions of their world.

Stage 1 Overview

The first stage of your assignment is to create a graphics display using Qt Creator (whichever version works but a recent version if possible!). Remember you must demo your code on the lab machines.

- Configuration file in a format that we DON'T specify — **NOT QSettings**. Use a TEXT file format. Call it what you want.
- The background must move at a constant speed. The stickman remains stationary until stage 2.
- The stickman can be created in four sizes defined as words within the config file.
- You will select a coding style to use, and specify it in the accompanying README.txt

Style I

Your specified style may be as simple as referring to an existing C++ style guide, or you may specify your own variations. Writing your own complete style guide would be far too excessive for an assignment.
e.g. The style for this project is based on the Google C++ style guide (link) with the following variations:

- The base indentation is 4 spaces
- System header includes come before local header includes.

You are expected to follow your own style guide (as will anyone extending your project).

Config file I

The Configuration file stores:

- The size of the stickman as a string. The options are: tiny, normal, large, and giant. (These will be used in Stage 3)
- Stickman's starting position as an x coordinate
- Stickman's starting velocity
- Background image
- Anything else you want, that's not specified in later stages.

Marking guidelines I

50%:

- it must compile and run on the lab machines;
- sensible / appropriate OO design.
- it must read the configuration file;
- stickman has four sizes
- the background moves not the stickman

Marking guidelines II

15%:

- appropriate use of a *creational* design pattern, with comments to explain its use;
- sensible error checking, e.g., for missing or incorrect configuration files;
- clear code using meaningful variable and method names as well as informative commenting and documentation;

Marking guidelines III

10%:

- memory, and other resources, are handled efficiently;
- clear code structure, that will be simple to maintain and extend;

Marking guidelines IV

The final 15%:

- Extensions to the functionality that are simple and effective, **using only Qt libraries and what you write**: e.g.,
  - sound effects,
  - ability to change colours and position and size of the stickman from within the program and then save the changes to a new configuration file,
  - pause and resume play (must not use the space bar or left and right keys, they are reserved for Stage 2 and 3);
  - animating the stickman (giving them a unicycle to ride perhaps)
- Documention of the extensions. e.g. README.txt to describe the features, and inline comments for developers to understand them.

5% penalty for each feature implemented from later stages.

Stage 2

What's required for Stage 2

1. One to two A4 pages typed review of the Stage 1 code you received. Portable Document Format (pdf) only. This component is worth 5%.
2. Extensions of the Stage 1 code. This component is worth 10%.

Marking guidelines I

Stage 2 is worth 15% to your final grade.
The first part is 5% and is a one-to-two page review of the code you received.
Don't skimp on the writing.
Illustrate using standard UML if you believe it will help.

Marking guidelines II

Your review will have 5 sections:

Documentation: how well was the code described, either as in-code comments or if there was separate documentation?

Extensibility: how well designed was the code for the extensions that you hope to make to it?

Design: what design patterns did the code use? Comment on whether you think they were good or poor choices, and justify your comments.

Implementation: was the coding well done? What would you have done differently? What was good about the implementation?

Style: comment on the style of the code. Were names, layout, code clichés consistent?

If you have studied the code thoroughly the above points can easily be made.

Marking guidelines III

Marks will be *removed* for

- poor spelling, grammar and/or punctuation;
- bad organisation;
- unprofessional terminology such as “the previous guy was an idiot” or “this is rubbish coding”;
- incorrect format submission (-1 mark penalty each time).

Extending Stage 1 I

Obstacles:

- Obstacles are initially laid out as a sequence according to the configuration file. Obstacles are rectangular and their position is based on the y coordinate.
- Obstacles must have configurable size and spacing between them.
- The obstacles should move at the same rate as the background.
- Once the sequence of obstacles defined in the config file finishes it should restart from the beginning (the game never ends).

Extending Stage 1 II

Movement:

- When the stickman hits an obstacle the background and obstacles should stop moving.
- The stickman should be able to jump over obstacles using space bar. Once they have jumped above an obstacle, the obstacles and background should resume moving.
- A test mode exists that replaces the graphical interface with a test interface for automated testing. The tests will include that collisions stop movement, and that jumping over an obstacle resumes movement.

Marking scheme I

The programming part is worth 10% for Stage 2.
50%:

- it must compile and run on the lab machines;
- Stage 1 behaviour, and interface, must be preserved;
- appropriate use of a *structural* design pattern;
- it must correctly use the configuration file to place both the stickman and the obstacles on the screen;
- crash safety: the program will not crash given invalid configuration files;

Marking scheme II

15%:

- intersection code works as expected and is extensible
- clear code using meaningful variable and method names as well as informative commenting and documentation;

Marking scheme III

10%:

- correctly deals with overlapping items: later obstacles cannot be placed over existing obstacles (and should be abandoned)
- different coloured obstacles as specified in the config file;

Marking scheme IV

15%, more sophisticated extensions:

- obstacles which move along the y axis at a specified rate
- double jump
- obstacles can have different shapes, as specified in the config file.

Discuss your extension idea(s) with your tutor before you go ahead!

Penalty of 1 mark if you go on and implement the next stage.

Stage 3

For your extension of Stage 2 : Its finally a game

- Just code: extend Stage 2 to complete the Flying Stickman game. This will include:
  1. Adding user control for the rate at which the character moves (they don't move on their own any more). This will allow you to move both backwards and forwards.
  2. The ability to lose. Crashing into obstacles now will result in the loss of a life.
  3. The ability to win. The obstacles should no longer repeat, rather there should be checkpoints to take you to new levels and then to finally win.
  4. An ability to provide a score to the user.
  5. Finally there should be powerups. The powerups should transform the stickman into their four personas each which should provide a possible advantage.

Marking Scheme

50%:

- it must compile and run in the lab (but this time you may use your laptop)
- preserve the previous two stage's functionality
- appropriate use of a *behavioural* design pattern
- clear code using meaningful variable and method names as well as informative commenting and documentation
- the stickman must have a configurable number of lives
- When the stickman intersects with an obstacle they lose a life and restart the level
- a score must be displayed, showing how well the player is doing (this could be as simple as a timer or as complex as you like)

Credit

15%:

- user control over the stickman. The left and right keys will control the character which will then in turn move the background and the obstacles
- there must be levels (at least 2) and therefore must be checkpoints that you can reach in each level to move you on to the next
- a *memory efficient* design;

Distinction

10%:

- include power-ups: the power-ups should alternate the stickman between their four states. Tiny mode should permit you to walk underneath some obstacles, large should permit you to jump higher than normal mode, and giant mode should make you immune to obstacles that is they explode on impact;
- a sensible testing framework for your code[3];

[3] Note: that doesn't mean you don't have to test your code before this!

High Distinction

15%, add some cool extensions, such as:

- further power-up
- infinite levels
- a scoreboard that persists between games!
- more things with exclamation marks!!

Make sure you discuss your extension idea(s) with your tutor before you go ahead with them.