Computer Science

CompSci 101
– ASSIGNMENT FOUR –

It's up to

YOU

- **Due:** **3:00pm, Wednesday 30th May, 2007**
- **Worth:** **4% of your final mark**

For this assignment, you need to implement a simple computer game – and the details are up to you.

There are a number of minimum requirements that you must satisfy in order to be awarded the marks, however most of the decision making lies in your hands.

You need to design and implement a simple computer game. The game must be animated, and there must be a well-defined goal that the person playing the game is trying to achieve.

The game must be controlled using the keys on the keyboard, and instructions for playing the game should be clearly displayed when the program runs.

This handout describes the minimum requirements for your game, and includes the marking schedule that will be used to assess your program.

REFERENCES:

- The resources for this assignment can be found at the following URL:

  `http://www.cs.auckland.ac.nz/compsci101s1c/assignments/`

AIMS OF THE ASSIGNMENT:

- to correctly handle different types of events
- to deal with animation and graphics
- to process arrays of objects
- to organise code into appropriate classes

WHAT YOU HAVE TO DO:

Read this handout carefully to make sure you satisfy the minimum requirements for this assignment. The rest is up to you…

Style

All of the code in every source file you submit must be indented correctly, using tabs only, according to the standard code conventions.

```
public void myMethod() {
	statement;
	if (condition) {
		statement;
	}
	statement;
}
```

Every source file (except A4App and A4JFrame) you submit must begin with a comment describing the purpose of that source file, or what kind of object is represented by the class.

```
/*
This program implements a game
in which ....
*/
public class A4JPanel {
	.... ;
}
```

All instance variables in every class must be declared `private`.

```
public class Robot {
	private int .... ;
	private String .... ;
}
```

All variable identifiers should adhere to the standard naming conventions and should be descriptive. The name of the variable should indicate what kind of information it stores.

```
Car userControlledCar;
Car enemyCar;
```

Correctness

Title screen

You need to think of a name for your game.

When the program first starts, a title screen should appear which clearly displays the name of your game and your login name. This title screen should also include a prompt informing the user of how to clear the title screen and continue with the game. In the example on the right, the user is prompted to press any key on the keyboard to start.

The size of the window should be approximately 500 pixels wide and 500 pixels high, although feel free to change this if you think it improves your game. It is advisable not to make the window too large. The **maximum** allowed size of your window is 800 (width) x 600 (height).

How to play

When your program starts, the person playing the game will want to know how to play. For example, they will want to know which keys on the keyboard to use to control the objects in the game, and what the objective of the game is.

You must display this information to the user **before** the game starts. One option is to include this information with the title screen itself as shown in the example to the right.

Another option would be to have a separate screen appear once the title screen has been cleared which lists these instructions.

You can format or position these instructions in any way that you want as long as the instructions specify which keys to use in the game, and what the basic goal is when playing the game.

Game screen

When the user clears the title screen and instructions (for example, by pressing a key on the keyboard) the main game screen should appear. The initial layout of this screen will depend entirely on the game you have designed and so no example is shown here.

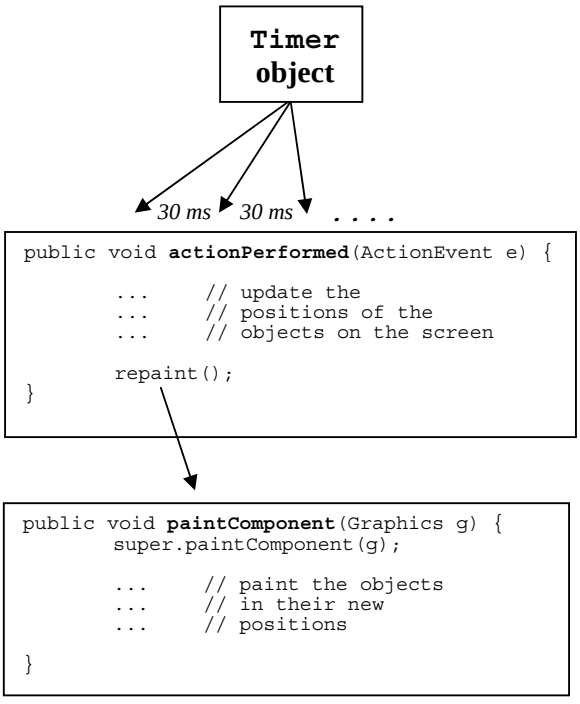
It is important that initially, any objects appearing on this game screen must be **stationary** – nothing should move. The animation for the game should only begin after the user has pressed one of the keys on the keyboard.

Animation

There must be at least one object (although there may be several) in your game that is animated, and therefore moves automatically. You must use the `Timer` object to perform this animation (as shown in the diagram on the right).

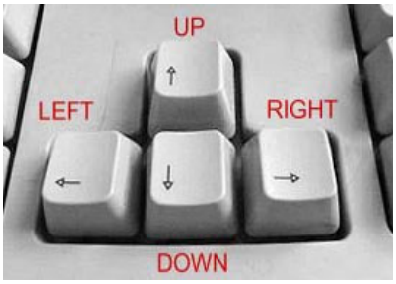
Remember, the animation of objects in your game should only begin once the user presses a key on the keyboard.

The directions and speeds with which the objects move depend entirely on your game.



Keyboard control

The user must be able to control the action in your game using the keys on the keyboard. It is likely that you will allow the user to interact with the game via the arrow keys, as shown below:



However, you may like to use another set of keys to control the action.

Of course make sure that you inform the user, when you display the instructions for the game, of which keys to use.

Goal

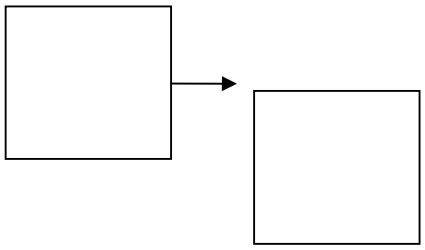
There must be some objective to your game. This objective, or goal, would have been displayed when the instructions describing how to play were shown.

You need to implement your program so that when the user plays your game they will be attempting to achieve this goal.

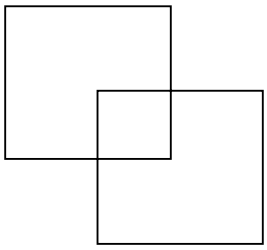
The goal itself can be very simple and may not even describe how to win the game. Your game may, in fact, be impossible to win. Instead, the player may simply have to play for as long as possible until they (inevitably) lose. Regardless of what you choose, there must be some objective for the player to attempt to achieve, and there must be some way the game ends either when the objective is achieved or when the player loses.

Collisions

Your game must consist of some objects for which it is necessary that you test whether the objects have collided. If the objects in your game are represented by squares or rectangles, then you can easily use the position of the rectangle to test for an intersection:



an object in your game may be moving towards another object



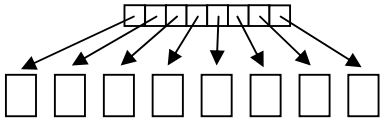
you need to test if the objects have collided, and if so, your program should react in some way (perhaps one of the objects will disappear)

User defined class

One of the aims of this assignment is to give you practice defining your own classes. You must define at least one class that represents an object (or several objects) that appears on the screen when the program runs. Because this class represents an object that appears on the screen, the instance variables for this class must (amongst other things) store the position of the object.

Array of objects

Another of the aims of this assignment is to give you practice processing arrays of objects. You need to use an array to store a collection of objects which are used in your game. The type of the objects in the array is up to you – it could be an array of some predefined Java class or of a class that you have defined yourself.



End of game

You need to implement some test to determine when the game is over. This may occur when the user has won the game, and achieved the objective, or it may occur when the player loses the game by failing to achieve the objective. This will depend entirely on the type of game you choose to implement, but either way, there must be some way for the game to end.

When the game is over, all animation must stop. The game screen should freeze showing the stationary objects on the screen at the time the game ended. The user must **not be able to move any objects** on the screen by pressing the keys on the keyboard. There should be **no way for the user to restart the action** in the game from the point where the game ended.

Final Notes

- Do not import any sounds or images into your game. Any drawing must be done using the primitive drawing operations available in the `Graphics` class.
- Your assignment idea should be different to that of other people in the class.
- Do not worry about the quality of the graphics in your game – there are no marks allocated for artistic ability.

SUMMARY OF SUBMISSION INSTRUCTIONS:

- You must submit **every** source file that your program requires. At the **very least**, you must submit:
    - `A4App.java` – your application class for running the program (this has been provided for you)
    - `A4JFrame.java` – the JFrame class (this has also been provided for you)
    - `A4JPanel.java` – this will process all of the event handling and control the animation for your game
    - And you must submit at least one other class that represents some object or objects that appear on the screen when the program runs

DO NOT SUBMIT SOMEONE ELSE'S WORK:

- Do not *under any circumstances* copy anyone else's work – this will be penalised heavily.
- Do not *under any circumstances* give a copy of your work to someone else.
- The Computer Science department uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken.

ASSIGNMENT FOUR COMPETITION

If you structure your code well, you may find that it is easy to extend your basic assignment into a game that is much more playable. You should focus on adding rules or challenges to the basic game which make it addictive and fun to play.

How to enter

If you would like to take part, then email your game to **paul@cs.auckland.ac.nz**

- The subject of the email message must be: "**A4 Competition Entry 2007**"
- The **body of the email** must include your **full name**, and your **login name**
- The **body of the email** must also state **which lecture stream** you will be attending on Thursday, 31st May (there will be one prize for each lecture stream – see the "Prizes" section)
- All of your Java **source files** must be archived into a single `.zip` file which must be attached to the email.

Deadline

Your competition entry must be received by **3:00pm, Tuesday 29th May**. Note, this is one day **before** the main assignment is due to allow adequate time for judging. **No submissions** will be accepted after this time.

Prizes

The prizes for the competition will be sponsored by Microsoft. One prize will be awarded in the 10:00am lecture stream, and another prize will be awarded in the 4:00pm lecture stream.

You do not need to submit the "best" game to win the prize. Once all the entries are received, they will be judged based on the criteria described later. A small number of finalists will be selected (possibly 8), and the prize will be awarded **randomly** to one of the finalists.

Judging

The two most important criteria for judging the entries are as follows:

- playability – ideally, your game should be addictive enough and challenging enough to make the user want to play it again and again
- uniqueness – rather than writing a clone of a well-known game, it would be preferable to design something unique

Demonstration

The best entries will be demonstrated to the class during the lecture on Thursday, 31st May. This will also be when the prizes are awarded.

Eligibility

As with the assignment, you must not use any imported graphics or sound files. Not only does this make the judging process much fairer, it also encourages you to concentrate on improving the rules of your game to make it more playable.

Obviously you must be an enrolled CompSci 101 student, and you must not collaborate with anyone in writing the code – it must be all your own work.

Paul Denny
May 2007

MARKING SCHEDULE

Style

Examine each of the source files submitted and review the following style categories:

Indentation

0 marks	**Inconsistent** – there is at least one place in at least one of the source files where the lines of code are not correctly indented according to the standard code conventions. Tabs must be used rather than spaces for the indentation
1 mark	**Perfect** – all of the code in each source file is indented correctly according to the standard code conventions

Comments

0 marks	**None** – there is no comment at the top of the source files describing the purpose of each class
1 mark	**Poor** – every source file (except A4App and A4JFrame) does begin with a comment, however in at least one of the source files, the comment appearing at the top of the file contains spelling mistakes or uses language which is unprofessional
2 marks	**Perfect** – a comment appears at the top of every source file (except A4App and A4JFrame) which clearly describes the purpose of that class

Private instance variables

0 marks	**Poor** – there are several instance variables (variables that are declared in a class outside of any method definition) which are not declared using the `private` access modifier.
1 mark	**Good** – there is at least one instance variable in at least one of the source files which is not declared using the `private` access modifier
2 marks	**Perfect** – every instance variable in every source file is correctly declared using the access modifier `private`

Descriptive variable names

0 marks	**Poor** – most classes contain at least one variable which has a name that does not describe the information which is stored in the variable. If it is not possible to figure out what the variable is used to store simply by looking at the name, then the name is not adequate
1 mark	**Good** – the names chosen for all (or nearly all) variables describe the information which is stored in the variable

Provide additional comments on the style of the source code

Correctness

Compile the program and run it.

Title screen

0 marks	**Compile error or no title screen** – the code does not compile, or the program does not display the title of the game on the screen when it initially runs
1 mark	**Title screen does not clear** – the program runs and the title of the game is displayed, but this title screen cannot be made to disappear by the user
2 marks	**Perfect** – when the program runs, the title of the game appears on the screen, and the user is able to clear this title screen and display the game screen

How to play

0 marks	**Compile error or no instructions** – the code does not compile, or the program does not display any instructions on how to play
1 mark	**Vague instructions** – the program runs and instructions on how to play appear somewhere (this may be on the title screen itself, or it may be when the title screen disappears), but these instructions are either unclear, or do not contain all of the necessary information (which keys to press to control the game, the goal of the game, etc.)
2 marks	**Perfect** – when the program runs, instructions on how to play appear (either on the title screen itself, or after the title screen) and these instructions clearly specify which keys control the game, and what the goal of the game is

Game screen

0 marks	**Compile error or no game screen** – the code does not compile, or the code compiles but no game screen appears
1 mark	**Not stationary** – once the title screen and instructions on how to play have been cleared by pressing the mouse button, the game screen appears, however some objects are not stationary to begin with. Any animation should only start once the first key is pressed
2 marks	**Perfect** – once the title screen and instructions on how to play have been cleared, the game screen appears and every object is initially stationary

Animation

0 marks	**Compile error or no animation** – the code does not compile, or the code compiles but no animation occurs
2 marks	**Perfect** – the game consists of at least one object which has animated movement. The animated object moves automatically (i.e. does not require multiple key presses or a key to be held down), and the animation is performed using the `Timer` object

Keyboard control

0 marks	**Compile error or no keyboard control** – the code does not compile, or the code compiles but the user is not able to control the game using the keys on the keyboard
1 mark	**Perfect** – exactly what sort of things the user does in the game depends entirely on the game itself, however the action is controlled by the user pressing certain keys on the keyboard

Goal

0 marks	**Compile error or no goal** – the code does not compile, or the goal of the game has not been implemented
1 mark	**Obvious errors** – the goal of the game, which was described in the instructions on how to play, has not been implemented perfectly. There are some obvious errors in the way the program behaves compared to how it is supposed to behave
2 marks	**Good** – a good effort has been made to implement the goal of the game, which was described in the instructions on how to play, but there are some minor errors with the implementation
3 marks	**Perfect** – the goal of the game, which was described in the instructions on how to play, has been implemented perfectly and there are no noticeable errors in the implementation

Collisions

0 marks	**Compile error or no collision testing** – the code does not compile, or the code compiles but there is no testing for any type of collisions between objects
1 mark	**Good** – some effort has been made to deal with collisions between certain types of objects, however there is at least one noticeable error where the collision of two objects is not handled correctly
2 marks	**Perfect** – collisions between certain types of objects has been implemented perfectly

User defined class

0 marks	**Incorrect** – there is no user defined class submitted as part of the program
1 mark	**Not used correctly** – there is at least one user defined class submitted, however this class is not used to represent an object (or several objects) that appear on the screen when the program runs
2 marks	**Perfect** – there is at least one user defined class submitted and this class is used to represent an object (or several objects) that appear on the screen when the program executes. The instance variables in this class must store information about the position of the object (or objects)

Array of objects

0 marks	**Incorrect** – there is no array of objects used in the program
1 mark	**Perfect** – there is at least one array of objects that is used when the program is run. The *type* of the elements in this array can be any object type (either a predefined Java class or a user defined class)

End of game

0 marks	**Incorrect** – it does not seem to be possible for the game to end, or the animation does not freeze correctly when the game finishes
1 mark	**Good** – when the game should end, the animation correctly freezes, however the animation can be restarted and the player can continue from where the game finished by pressing keys on the keyboard
2 marks	**Perfect** – when the game ends, the animation freezes and there is no way for the player to continue from the point where the game finished. There may or may not been an option for the player to restart the game from the beginning.

Provide additional comments on the correctness of the program