National University of Singapore
School of Computing
CS3217: Software Engineering on Modern Application Platforms
AY2011/2012, Semester 2

Problem Set 5: **Putting It All Together**

Issue date: 12 February 2012
Due date: **27 February 2012 23:59**

Introduction

In this problem set, your task is to complete the implementation of **Huff and Puff** by integrating the modules you implemented in Problem Sets 3 & 4.

In this assignment you will see your efforts over the past few weeks to completion. You have three key tasks in this assignment: (i) extend your implementation of the physics engine to support circles, (ii) integrating the physics engine with the MVC-based GUI from Problem Set 3, and (iii) tying up the loose ends to complete the game.

The objective of this assignment is to help you better understand design and integration in software engineering. If your did your past two problem sets very well and debugged your code thoroughly, the integration should not be too difficult; on the other hand, if your previous problem sets still have bugs, the bugs will come back to haunt you and you would probably want to debug them before attempting to integrate. How you choose to do this assignment is really up to you. Our goal here is to provide you with an opportunity to learn.

Note that while you are required to achieve certain technical requirements in the Problem Set, you are given the latitude to decide on the actual rules of the game you implement. As long as the final game involves projectiles, blocks reacting differently to the projectiles depending on types, you are pretty much allowed to do anything you want. You don't even need to use the graphical images we provide and and you can change the background.

Reminder: Please read the entire assignment before starting.

There are two zip files that come with this problem set:

- `ps05-images-required.zip`, which contains the image files for the required specifications;
- `ps05-images-extra.zip`, which contains various sprites for extra-features that you might wish to implement.

Please make sure you have both of them before starting.

Important Note:	Students who had a lot of trouble implementing the game engine in Problem Set 4 (and whose game engine is still buggy) can alternatively choose to use an open-source 2D engine for this assignment.

Problem 1: What is YOUR game gonna be?

Unlike your previous problem sets, the grading of the assignment is divided into two components: satisfying the technical requirements (70%) and "coolness" factor (30%). Before you get too stressed, note that satisfying the technical requirements well is sufficient for you to get a solid A-grade in CS3217. The 30% "coolness" factor element in this problem set is designed to provide you with the chance to show off your creativity and to earn an A+-grade (or make up for mistakes in past problem sets and/or earn a free iPad 3).

The following are the conditions that your game must satisfy:

- It involves a physics engine and both round and rectangular objects colliding and bouncing.
- It involves one or more projectile of sorts.
- It involves some entities (possibly pigs, possibly not) being destroyed.
- It allows for the creation, saving and loading of multiple "levels" or level-equivalents.

As long as your game satisfies the above conditions, you are free to do pretty much anything you want. You are free to replace the background and provided images, if so desire.

First, read through this entire problem set to understand the technical requirements for the problem. Next, describe the rules of your game and how the game works in a file called `design.txt`.

Note:	You might wish to have some "pre-programmed" levels for your game, instead of expecting the user to "create levels".

Problem 2: Implementing the Puff *(100 points)*

In Problem Set 3, you implemented the level designer of the game. The user can place objects from the palette in the game, rotate and resize them, save and load levels. It is time to implement the actual game. The game starts when the user presses the **START** button (though you are free to change the user interface to something else that accomplishes the same). This will trigger several things: the physics engine will be activated, game objects will respond to new behaviour, and animations will be enabled.

In this part of the assignment, you have to extend your previous works with two new components. Describe your implementation of the following:

a. **Wolf Breath Direction & Power**. You should extend the project from Problem Set 3 to add a new game object, the wolf breath. When the game starts, the user selects the angle and the power for the projectile. We have provided you with the art assets, but you need to decide how to allow the user to specify the angle and the power for the wolf breath and implement it accordingly. In the provided zip files, you can find the following images which might be helpful:

   - `direction-arrow.png`, a sprite that indicates the direction of the projectile,
   - `direction-arrow-selected.png`, a sprite that can be used when the user modifies the projectile direction,
   - `direction-degree.png`, a sprite that shows the angle of fire for the projectile,
   - `breath-bar.png`, a sprite that shows the power of the breath,
   - `windblow.png`, an animation sprite containing 4 frames that are used for the projectile. Please note that the wolf breath is animated since it is fired until it is destroyed.

   Explain in the file `design.txt` how your game handles user input for the angle and power of the projectile and implement it. Describe how you integrated the power of wolf breath with the physics engine. Explain why you chose this method over alternatives. (*30 points*)

b. **Wolf Breath**. Next, you need to extend the game engine from problem set 4 to include circle-rectangle interaction. In the previous assignment, there were only rectangle-rectangle interactions. In the game, the projectile is represented as a circle, and thus you need to extend the game engine to handle the collisions between the wolf breath and blocks. Describe in the file `design.txt` how you would extend your original physics engine to support circles. (*20 points*)

   Using an Open Source Physics Engine. Students who decide to use an open-source physics engine won't get to extend their physics engine for this question. In place of the circle-rectangle interaction, you will need to do the following:

   - Explain why you decided to use an open source physics engine.
   - Explain how the open source engine works and/or is designed and how you integrate your project with the engine.

- Highlight how the design of the open source engine differs from the the physics engine done in PS 4.

c. **Object-Object Interactions.**Finally, you need to implement different behaviours for the collisions in the game. The collision between wolf breath and a *straw* block results in the destruction of the *straw* block, while the power of the wolf breath is halved. Bonus points will be given for a realistic implementation of this type of collision, where the breath continues on the same trajectory, but with less impact power for the next collision. The collision between wolf breath and any other type of block results in the destruction of the game breath and an impulse on the game block, computed by the physics engine. Please note that the different types of blocks should have different mass, and collisions should behave differently according to what blocks collide. Your game is allowed to have slightly different behaviours as long as you can demonstrate that you are able to implement at least three different interactions between game objects. Describe in the file `design.txt` your three different interacts and explain your general strategy for implementing different types of behaviours between objects. Explain why your strategy is the best among alternatives. (*30 points*)

d. **Starting the Game**. After adding the new game object (wolf breath), and implementing the circle-rectangle interaction, it's time to put them all together. Integrate the physics engine in the game project, and activate it when the user pushes the *START* button. Object interactions should then be handled according to the rules of the game and the physics engine. You are free to choose *NSTimer*, *Central Dispatch* or *NSThread* to implement the game loop as long as your application runs smoothly and responsively. Describe in the file `design.txt` how your design allowed the integration of the physics engine. Indicate which approach you used to run the game loop. Explain the advantages and disadvantages of your approach and alternative approaches. (*15 points*)

e. **Module-Dependency Diagram.** Draw the MDD for the basic implementation of your game (i.e. you do not need to include the bells and whistles from Problem 4). Save you diagram as `MDD.png` and include it in your repository for submission. (*5 points*)

Problem 3: Integration Testing *(30 points)*

Testing is an integral part of software engineering. For the final game, you are supposed to implement a large number of features and it is important for you to test your final code to make sure that the game meets the stated requirements. The way to do this is to start from a hierarchy and then break down into smaller and more specific cases. For example:

- Black-box testing
  - Test level designer
    - ···
  - Test start game

    * ···
  - Test wolf breath
    * ···
  - Test physics engine
    * ···
  - ···

- Glass-box testing
  - ···
    * ···

Please describe your testing strategy in `design.txt`. The testing strategy should also include the testing strategies for Problem Sets 3 and 4, since these are components of the main program. If you did your testing perfectly in the previous problem sets, you only need to replicate what you did earlier; if you didn't do so well, this is where you show that you've learnt something and updated the tests.

Of course, you should test the game as you have described instead of just listing down what you think you ought to test! You might want to let your friends try out your game and thereby help you test it too. :-)

Problem 4: The Bells & Whistles *(60 points)*

With a good design and good implementation for problem sets 3 and 4, extensions and integration should not take too much time. Spend your remaining time by finishing the game and adding extensions. The points in this part are subjective and take into consideration the general feeling of the entire game and will be awarded relative to the submissions by the rest of the class. The following is a list of possible improvements that you can consider implementing:

- There are two animations that you can add to the wolf using the `wolfs.png` sprite: inhale, and exhale.
- Animation for the destruction of the breath. 10-frames `wind-disperse.png` animation sprite file included in the archive;
- Adding game score. `font.png` and `score.png` sprites available;
- End-game screen.
- Pig animation. `pig2.png` and `pig-die-smoke.png` sprites available;
- Improve wolf animation: wolf sucks air when inhaling. `windsuck.png` animation sprite available;
- Wolf has a limited number of breaths available, after which it dies. `heart.png` and `wolfdie.png` sprites available;
- Different types of wolf breath. `windblow1.png`, `windblow2.png`, `windblow3.png` animation sprites included;

- Add trajectory animation to the wolf breath.

You are free to add your own features to show off your creativity. Please describe in the file `design.txt` all the extra-features and improvements and briefly describe how you modified your original design to implement each feature.

Problem 5: Final Reflection *(10 points)*

Note:	You should answer this question only **after** you have completed this problem set.

Comment on the original design of your MVC architecture and also on the design of your physics engine. Is it possible to further improve the design/architecture? If so, how? If not, why not?

Bonus Problem: Reflection (4 Bonus Points).

Please answer the following questions:

a. How many hours did you spend on each problem of this problem set?

b. In retrospect, what could you have done better to reduce the time you spent solving this problem set?

c. What could the CS3217 teaching staff have done better to improve your learning experience in this problem set?

d. Congratulations, you have completed all the Problem Sets for CS3217. To recap, the learning objectives of the various problem sets are as follows:

   (a) Problem Set 1: Introduction to XCode, Interface Builder and Objective-C
   (b) Problem Set 2: Specifications, rep invariants, and unit testing
   (c) Problem Set 3: MVC and GUI-programming in Objective-C.
   (d) Problem Set 4: Design and implementation of a full software system.
   (e) Problem Set 5: Integration of modules and integration testing.

   Yes, we admit that the learning curve is very steep, but we don't have a lot of time to get you up to speed. The real question is: do you feel that the problem sets have been well-designed to help you learn software engineering and Objective-C is a structured and effective way? Do you have any suggestions on how the problem sets can be improved?

Your answers to these questions should be appended at the end of `design.txt`.

Grading Scheme

In this module, you are being trained to become a good software engineer. The first and basic requirement is that your code must satisfy the requirements and be correct. Above and beyond correctness, you are required to write well-documented code.

For this problem set, we will be testing your code by compiling your app and uploading it to an iPad to make sure that the game works as expected. You should also probably upload the application to your iPad to make sure that you can play the game. We will be looking at the following:

- Your submission should adhere to the submission format.
- You have answered the questions satisfactorily in a concise manner.
- Your project should build without errors or warnings.
- Your project should run without crashing.
- You have implemented extra features.
- Your game is complete. By this, we mean that you can create and play a level, the wolf can fire, the objects move according to the physics engine, the game ends when the pig is destroyed (or when the wolf finishes the number of breaths), the score is displayed when the game ends.

Mode of Submission

The teaching staff will be grading your code directly on GitHub. You will be graded on the latest commit on the master branch before the deadline. Your solution for this Problem Set should be contained in a single directory called `ps05`, which should be inside the root directory of the private repository assigned to you. The `ps05` directory should contain **all** your project files. However, in order to keep the size of your submission small, you should omit the `build` subdirectory that contains the compiled binaries. In addition, your submission directory should contain your design explanation `design.txt`, a module dependency diagram `MDD.png` and a README file where you specify which implementation files contain your code.

Clarifications and questions related to this assignment may be directed to the IVLE Forum under the header “Problem Set 5: Putting It All Together”.

Good luck and have fun!