Technical Report

Why?

- So we can assess YOUR understanding of the solution or problem.
- To improve your understanding of the implementation/solution.
- Not replace comments IN the code!
- To enable a reader to reproduce your work.
- Prepares you for later work which will probably include writing technical reports of some description.

What?

- A detailed technical, objective description of of YOUR implementation/solution to a given problem.

What not!

- It is not a diary of how you arrived at your solution. We don't care how difficult you thought it was or how frustrated you were.

So what DO we put in a report?

Introduction

- A short description of your report.
- A short description of the problem.

“This report details the implementation of a reliable data transfer protocol sitting between the network-layer and the app-layer in the classic osi-stack architecture.”

Requirements

- Could be a sub-chapter of the Introduction, or a stand-alone chapter if it contains a lot
- Describe what is required of the solution.
- NOT what you needed to complete it; a computer, lots of time...

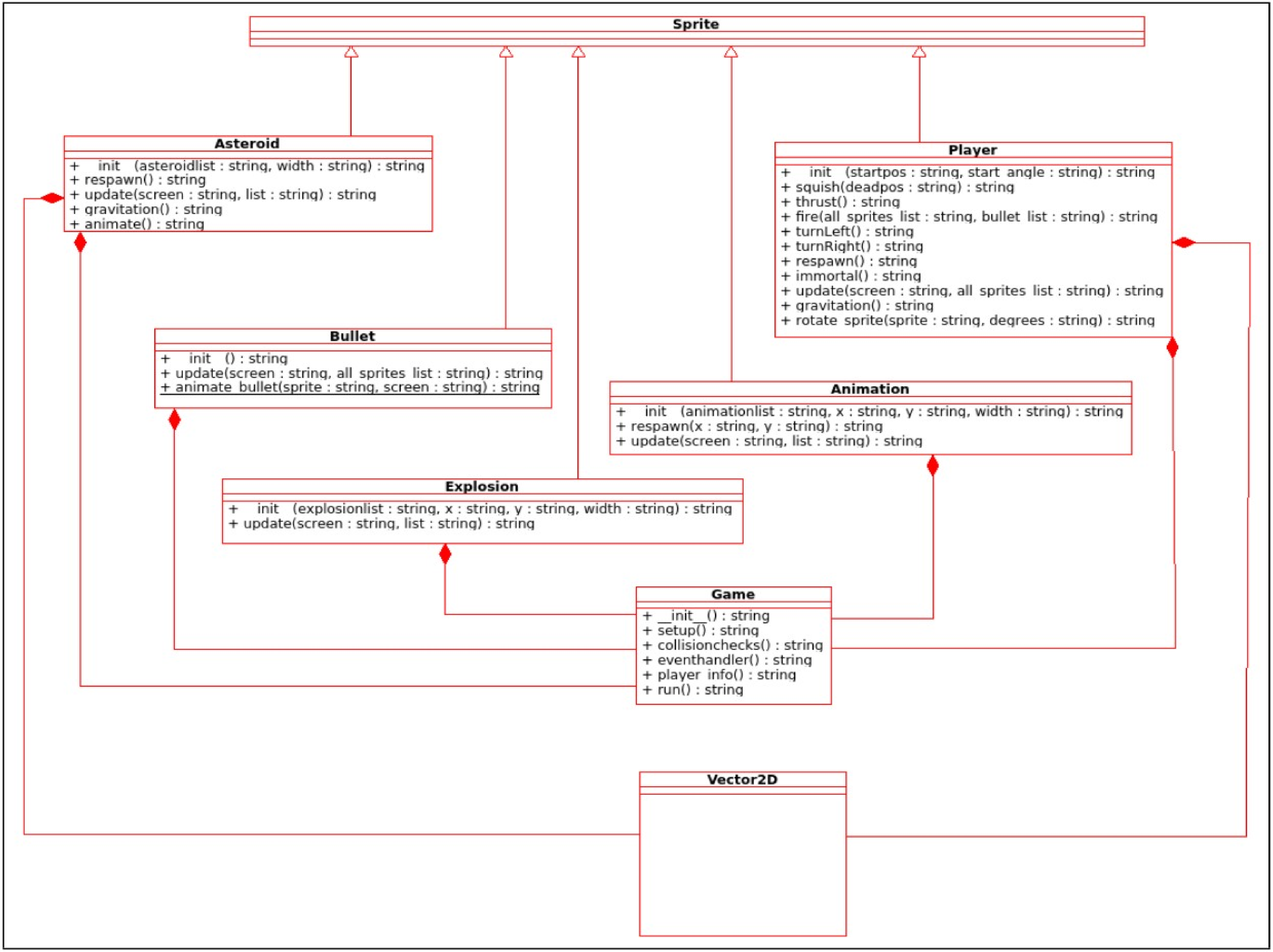
“...to implement a transport layer protocol capable of handling network delay, packet loss and data corruption.”

Technical/Theoretical Background

- Describe new concepts that you utilized in your solution.
- All new tools.
- Pygame?
- Objects and classes?
- Game loop?
- Etc?

Design

- NOT a description of how something looks! “I chose red bricks for my game...”
- It is a structural description of your implementation.
- Don't dive into long descriptions of code. Your implementations will soon feature 1000+ lines of code, just give me the basic overview!
- A good place to put your UML(class diagram)!


Sprite
Asteroid
+ init (asteroidlist : string, width : string) : string
+ respawn() : string
+ update(screen : string, list : string) : string
+ gravitation() : string
+ animate() : string
Player
+ init (startpos : string, start angle : string) : string
+ squish(deadpos : string) : string
+ thrust() : string
+ fire(all sprites list : string, bullet list : string) : string
+ turnLeft() : string
+ turnRight() : string
+ respawn() : string
+ immortal() : string
+ update(screen : string, all sprites list : string) : string
+ gravitation() : string
+ rotate sprite(sprite : string, degrees : string) : string
Bullet
+ init () : string
+ update(screen : string, all sprites list : string) : string
+ animate bullet(sprite : string, screen : string) : string
Animation
+ init (animationlist : string, x : string, y : string, width : string) : string
+ respawn(x : string, y : string) : string
+ update(screen : string, list : string) : string
Explosion
+ init (explosionlist : string, x : string, y : string, width : string) : string
+ update(screen : string, list : string) : string
Game
+ __init__() : string
+ setup() : string
+ collisionchecks() : string
+ eventhandler() : string
+ player info() : string
+ run() : string
Vector2D

Implementation

- Implementation specific details.
- How you solved a particularly difficult problem
- Bugs you encountered and solved.
- Bugs you encountered and didn't solve. What do you think might be the cause?
- Try to describe your codes strengths as well as weaknesses.

Technical Details

- A sub-section of the Implementation chapter could be details concerning the environment.
- This might explain system specific bugs or clarify whys and hows of your implementation.

“Programming language:

Both createimage.c and bootblock.S were written in C and AT&T assembly respectively.

Editor:

Sublime Text Build 3126.

Environment:

Linux version 4.4.0-47-generic, running Linux Mint 17.3 Rosa

Testing:

Bochs x86 Emulator 2.6.8

Compiler:

gcc (Ubuntu 4.8.4-2ubuntu1 14.04.3) 4.8.4”

Discussion

- Discuss the strengths and weaknesses of your implementation. Why is your solution good? Why is it not? How would you improve the mentioned weaknesses?
- Explain your choices.
- Compare your solution to alternative ones(not your class mates!).
- This is where you show in depth understanding of the problem/solution.

Conclusion

- Argue why your solution meets the requirements.
- Again, NOT your feelings about the finished result.
- Lessons learned?
- Does everything work as expected?
- Short and concise!

References

- Any academic text MUST reference all external sources of information. Any journal, book, publication or website used must be referenced.
- Fail to reference and you are plagiarizing other peoples hard work and passing it off as your own.

Language

- Technical to the point of boring. Short. To the point.
- Write as though the reader is an expert in the programming language and tools you are using.
- Refer to your own specified requirements, NOT the assignment text!
- Never use personal nouns! (“I”, “We”, “You”…)
- Exceptions could be intro and conclusion, but a well structured sentence avoids it easily.

Figures

- A figure should tell the reader more than the same amount of space could in words.

Good use:

- Complement a complex explanation
- Graph relations between lots of data
- Show overall design
- Connections between classes

Bad use:

- To fill space and make your text seem longer(and thereby better?!)

Length?

- A long report is NOT necessarily a good one.
- ...neither is a short one for that matter…
- Try not to think about how long it MUST be.
- Rather, how long is it after you have written everything that in your opinion MUST be in it?

  Some of the best reports I have read were 2 pages long.

  Some of the worst were also 2 pages. And 16...