Team: “I swear this was working a minute ago…”

PHASE 3 Report “Coding phase”

By:

Jose A. Torres and Israel Mendoza
Initial decisions to make:

- What language are we going to use?
- Is the project going to be object oriented or just use functional programming?
- Who is going to code what?
- What are we trying to accomplish?
- How do we accomplish this within the given time?

These are the main questions that we had to ask ourselves as a group in order to get started on flatland defender. So we met up and started discussing what languages everyone knew and how comfortable everyone was with specific languages.

So when we met up and talked about what language we were going to actually use the first three that came up were: C#, C++, and plain old C
Team members who actually knew C#:
2 out of 4 (only half the team)

PROS:

- Easy to pick up and use
- It isn’t a complicated language, kind of like Java
- The statements are more natural and make sense when you are coding
- It is an easy object oriented language to program in
- It is usable on a wide variety of platforms
- There are built in “bounds checking” for arrays

CONS:

- Two people on the team have had no experience with C# so for them it wouldn’t just be programming a game; it would be programming a game in a language they have no experience with.
- C# was developed by Microsoft, nothing they do ever really works right.
- The two that know c# one has limited experience with it.
- New keywords
- Can’t just start programming in a main, have to do a couple of methods before being able to do anything.
- Implicit boxing
- No pointers in C#
- Feels less practical and more theoretical
Team members who actually know C (really well):
1 out of 4 (25 percent)

The lack of knowledge and the lack of classes was the reason that we quickly moved on from this language but here are the pros and cons.

**Pros:**

- Widely used programming language, so there are a lot of resources on the internet
- Low level language
- Programs are stand alone and not dependent on an IDE
- Fast running
- Very powerful, in terms of what it is capable of
- Simple, core language

**Cons:**

- Programs written in C can be difficult to port
- Not easy to learn, this language is not intuitive like others
- Not object oriented
- Programs have to be compiled separately for each platform
- No bounds checking whatsoever
- Programs can get really complicated really fast if not an experienced C programmer, even then might still get really complicated
- Some things in C are more complicated as a result of the way it was designed.
**C++:**

Team members who know C++:
4 out of 4 (100 percent)

**Pros:**

- It’s used everywhere (practically)
- It has object oriented features
- Programs are stand alone
- Easy to port to other platforms if strict C++ guidelines followed
- You could include libraries for added functionality
- Our team is relatively more familiar with this language
- It would be the easiest to work with

**Cons:**

- Not easy to just pickup and learn
- Size of programs, if complex can be large
- No bounds checking
- Non-specific programs could be difficult to port because of the libraries that they may depend on

**Decision:**

So after reviewing the top three languages we came up with and looking at our pros and cons lists, the choice was obvious since it had the fewest cons and we were all comfortable with it, we chose to do our game in a mix of C++ and C, since C++ is just a superset of C.
Object oriented vs. functional programming

For our game we had to make a choice which involved whether our game would be object oriented or not. We had this choice because in C++ you can choose whether or not you want to employ and take advantages of the features with classes. After a small discussion on how much easier it would be to just do object oriented we came to an almost immediate consensus and chose to do object oriented. The reasons included: The ability to create classes, the ability to have a class scope (use of the scope resolution operation for classes), function overloading, and the ability to use inheritance to derive classes from other classes such as a player class derived from the object class, as well as an enemy class derived from the same parent class, not to mention the object or entity class could also just be general in game objects if need be. It just made more sense to go with Object oriented programming on this particular project.
How we divided the work:

We divided the work in half, forming two teams of two for the general coding of the project and when it was crunch time we came together as a larger 4 man team to attempt to resolve issues that we were having.

When we got down to the actual programming at least on my sub-team of me and Israel, we did some extreme coding where we would take turns coding while the other supervised and rested. We also found it to not be the best strategy, as this would sometimes lead to the one resting/supervising to get sidetracked and then both teammates would get sidetracked. We felt that the two sub-groups worked best for our team since 2 of our teammates worked together and as such spent more time together. This made it easier to actually get stuff done as myself and Israel have a similar schedule we were also able to get together and get stuff done. We used github.com to manage our files and keep everything up to date. Whenever someone made changes we just had to use the “git pull” command that Brandon showed us in class to pull the latest changes to our local copies of the repositories. One thing that was interesting to deal with was when someone “broke the build”, we had to undo changes that person committed and just backtrack logically to make sure we got it back to the exact place that the project was broken. Then we had to try and figure out why that particular thing had broken the program/game.
Coding: On a time budget

We have only ever had a short interval of time to get this project together, and as such everything has been rushed. Since the four of us have very different schedules and as such one of the biggest things to overcome was the fact that we could never meet and code our game together as a group. So we basically did two extreme programming teams in order to code as quickly and efficiently as possible. Extreme programming was our solution to coding the game within the limited amount of time.
Extreme programming is a legitimate software developing method that was created to get a program up and running as quickly as humanly possible. With the focus being on functionality first aesthetics later with the goal being to first and foremost get the application working. The coding is usually done in groups of two, or pairs so that each programmer can code and rest as needed, in order to get as much done as possible within a shorter amount of time. Optimization of the software’s code doesn’t actually happen until the end. This programming method is derived from the agile programming method, it is a form of it. It also attempts to be as simple as possible, in order to not have unreadable, convoluted, code that can never be worked on again by anyone else because they can’t figure out what each thing does or why you made certain variables. One problem that we encountered while doing extreme programming was when both programmers would get tired and then easily distracted and start screwing around on the internet, instead of developing the code and working to get the application up and running. Other than that extreme programming was interesting and fun.