This document contains some reflections on my (Quang Luong’s) Dice Invader solution.

**State of game:** Game works according to specification with a bunch of extras. Ship at bottom fires rockets upwards at enemies while they randomly fire bombs downwards. But to mix it up, I put in 6 player ships. (Number of ships can be set at game over screen with left and right, and space restarts game.)

**Project structure:** I tried to keep the style like the one in the given files to keep things more consistent. I went for a Model-View-Controller (MVC) approach for organizing things, so GameObjects are the Models, the GameObjectControllers (and children) are controllers, GameMain also has controller functionality as it handles the game loop, and uses the library to draw the models. As much encapsulation as possible was strived for, while also maintaining easily readable code. (Code is mostly self-documenting.) Pointers were intentionally used as little as possible to keep code more intuitive and were seen as a last resort, instead using mainly object instances which also handles memory automatically.

**Flexibility:**  An initial approach was all for simplicity, having all logic directly in the main game loop and using simple calculations. This minimized function calls and types and thus was more efficient. After that, a new approach aimed to make the code as flexible as possible, adding 2D geometry (Vector2) to allow more freedom in movement (although only 4 directions are used, to maintain gameplay mechanics). At the cost of more function calls and calculations.

GameObjects are independent of anything, making them very reusable. GameObjectController is also very generic, being a template class, which could easily be repurposed. The GameObjectController subclasses ( ShipController, EnemyController etc) and even GameMain contain logic specific for the game. Everything related to the Dice Invaders library is only used in GameMain where the core logic also is.

**Container of choice:** std::vector became the natural STL container of choice used in this solution due to its simplicity. Operations used are mostly (if not all) constant time. No time is wasted sorting. No time is wasted looking up. The solution iterates through the vector from beginning to end and applies logic to every element in order.

**Collision:** Every type that can collide is checked against the types they can collide with. (Rockets collide with ~~bombs and~~ aliens, bombs collide with player and enemies collide with player). This is simply done by calculating distance, which was intended to be the broad phase, but worked well visually and therefore didn’t need a more refined check, which would only be costlier. A narrow phase solution could have checked if the width and height (AABBs) overlapped anywhere.

**Memory use:** Only one of each ISprite is created. Every controller could have had a pointer reference to the associated sprite(s), but that would create a dependency for the DiceInvaders library which I tried to avoid.

The sprites (BMP images) are the costliest things in the game in terms of memory, just one image is 3x32x32=3kB (the first 3 being the color depth which is 24 bits = 3 bytes (RGB)) each totaling 15 kb, whereas variables take a few bytes each and std::vector can take around 30 bytes overhead. To compare, an “enemy army” would be the most memory consuming object in the game, consisting of 11x5=55 GameObjects (enemies), which have about 5 variables, using about 5 bytes each: 55\*5\*5 = 1375 bytes. Visual Studio profiler says it’s about 1.2k bytes.

Cheaper types are also preferred to the more detailed ones. int (often 4 bytes) before floats where possible. Unsigned wasn’t used to allow for flexibility with negative values, and maximum values don’t go beyond a couple thousands anyway, which is far from the maximum int values. Had focus been on maximum efficiency even smaller types like short int (often 2 bytes) can been used or even char (1 byte) or even bit manipulation where possible for bools, and small variables. From a bigger point of view, this game is so small that these small differences don’t make much of a noticeable difference, performance wise.