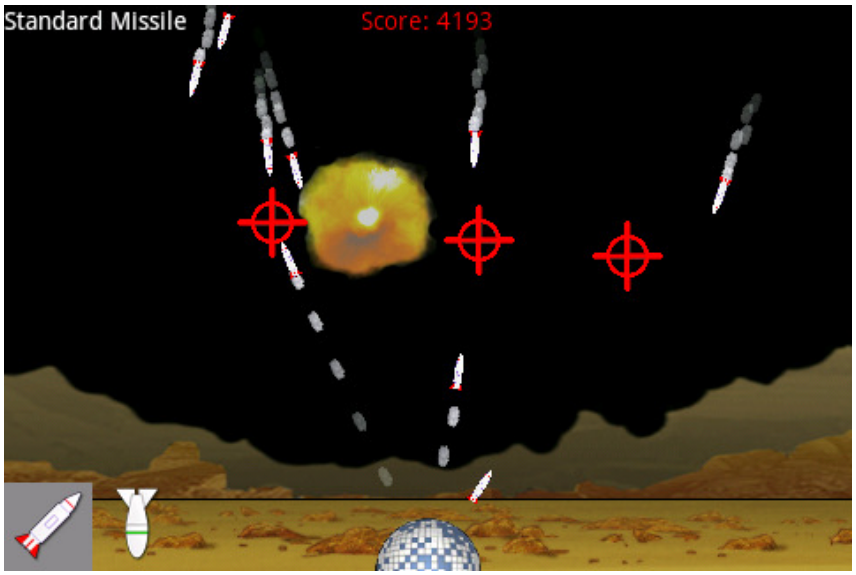
BOOM (Ballistic Ordinance & Orientation Manager)

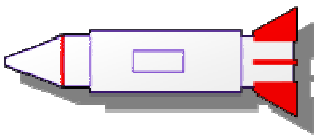
Boom is a spinoff of the classic arcade game Missile Command. The object is to defend your base, located in the bottom-center of the screen, from a horde of incoming enemy missiles. Most enemy missiles are fired in random directions and may not hit the player's base, but others are equipped with precision targeting to ensure a direct hit (and the destruction of the base).



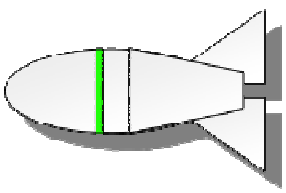
Players Instructions

For the purposes of this demo, the game difficulty starts on easy, and progressively gets harder as time goes on. Enemy missiles fly faster and are launched more frequently. The game continues until the player's base is destroyed. The goal is to get as high a score as possible.

Friendly Projectiles



The player may currently use two types of projectiles to destroy enemy missiles, a Red Interceptor Missile (right) and a Smart Shell (below). The currently selected projectile can be changed by touching the icons in the lower left corner of the screen. All projectiles launched originate from the player's base, and will travel in a straight line to the target point touched on the screen. Note that both types of missiles may collide with incoming enemy missiles along the way, causing them to explode prematurely.

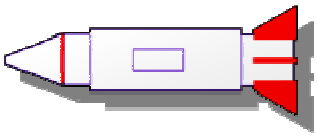


Interceptor Missiles will travel from the player's base to the point clicked on the screen, at which point they will explode. Smart Shells are similar to the Interceptor except they travel faster, have a smaller explosion radius, and are capable of destroying all both enemy missile types (more on that in the

Enemy Missile section). Launching a Smart Shell is slightly different than an Interceptor: instead of a single click, it takes two clicks. The first click causes the camera to zoom in on the point selected, and the second click launches the Smart Shell. This allows the Smart Shell to be precisely aimed, a necessity given its smaller explosion radius.

Enemy Missiles

Boom currently has two types of enemy missiles, the Red Destroyer and the Gold Seeker. Both missiles have random launch trajectories, allowing them to originate from anywhere at the top of the screen.



The Red Destroyers (right) are launched at varying velocities and have random targets on the ground, meaning that not every Destroyer will target the player's base. This allows the player to pick and choose their targets based on the threat they pose to their base, giving them the ability to destroy non-threat missiles for extra points. Destroyer missiles deal up to 10% of the base's maximum hit points, depending on the missile's proximity to the base at the time of the explosion. Destroyer missiles may be destroyed with any type of ammunition available to the player.

The Gold Seekers (left) are launched at the maximum velocity and always target the player's base. These missiles deal enough damage to instantly destroy the player's base, making them a top priority. Unlike the Destroyer, however, the Seeker may only be destroyed by the player's Smart Shells. Seekers initially have a small chance of launching, which gradually increases as the game goes on.

Scoring

A player's score is determined by the number of enemy projectiles destroyed and the accuracy of the detonation which destroys them. For example, a direct hit on a Red Destroyer results in the maximum amount of points plus a direct hit bonus. An enemy missile which is destroyed by an explosion gives a score which is based on the explosion's radius at the time it destroys enemy missile. Enemy projectiles destroyed with the Smart Shell have a double score multiplier, meaning they give double the points.

Requirements

Android SDK version m5 (tested with m5-rc15)
Android Emulator (**screen must be set to HVGA 480x320 Landscape**)

Developer Notes

The following is a summary of some of the thought process that drove the development of Boom as well as some of the issues we ran into along the way. Any currently known bugs included in the submitted version and references are also described. This information is intended for developers who have played Boom and are also familiar with the Android SDK.

Design Goals & Decisions

Fun – We were leaning towards making a game from the get go. Given the time constraints, team size, and familiarity with the platform (zero), we decided a simple, fast paced arcade game would be feasible, fun to code, and still provide a lot of room for experimentation and potential growth.

Simplicity – None of the Boom team has ever designed/coded a game before, so we used Missile Command[4] as a starting point and looked at ways of improving on the concept.

Speed – Given the fast paced nature of the game, the Android platform (and emulator) presented several technical challenges:

1. Frame Rate – The UI thread appears to have a 10 fps (frame per second) ceiling that proved unsuitable for our needs. After much experimenting with different rendering methods (OpenGL fixed point, OpenGL floating point, Canvas (UI thread), SurfaceView[3], etc), we choose to build a custom SurfaceView class and separate the UI and rendering threads.
2. Screen Size – This is an issue with many embedded systems projects. It also was more of an issue with our project because Missile Command typically benefits from a taller and wider the screen (the more launch space, the better). We decided to present the demo only in HVGA. We compensated for the shorter screen height using a camera with quick zooming and panning capability. We also slowed down the incoming projectiles a bit to give the user more reaction time.
3. Input Response – We capture the touchscreen events (mouse clicks in the case of the emulator) using the MotionEvent[6] class. Initially, the plan was to have the camera motion be a bit more flexible during zooming by allowing the user to drag their finger across the screen to pan and zoom simultaneously. This kills the frame rate, unfortunately. The problem is the UI thread hogs the CPU because it's constantly servicing the MotionEvent handler while the user has their finger (mouse click) held down.

Future Improvements, Plans, & Features

Camera – The camera (inherits the Android Camera[2] class) currently pans and zooms at a fixed velocity. This creates a bit of a "jerking" effect when moving short distances. Our design allows for different types of camera motion to be added by separating the Camera2D and CameraMotion2D classes. Future versions of the camera will allow for variable velocity and acceleration.

Accelerometer – When an advanced Android prototype becomes available, we'd like to experiment with a new method of panning using an accelerometer. This would allow the user to simply tilt the device in any direction to pan around the screen. Increasing the angle of tilt could also dynamically adjust the panning velocity (as discussed in the Camera section above).

Particle Effects – The current particle system[5] only has two effects, a smoke trail and an explosion. This will be expanded to include several different explosions, flame effects, and damage effects (debris, etc).

UI – Android has an incredibly flexible animation system that we plan on utilizing to polish the UI. Effects such as smooth motion, fading, transparency, and filters will be fully utilized to enhance the interface. The following additions are also planned:

1. Info Updates - To enhance game play, the UI will also be modified to report much more information during play than it currently does. Examples would include dynamically placed text messages that report score multipliers and the proximity of a close explosion or direct hit.
2. Reload Timer/Indicator - A progress bar will be added to the UI which reports the current progress of projectile reloading, and inform the user when a reload is complete.
3. Weapon Selection - The buttons used to select projectiles will be built into a slide out menu similar to what the Android emulator (m5) shows when selecting the "All" button. This will allow the menu to remain hidden when not used.

Weapons – The demo only has four projectile types at the moment. Obviously, we plan on adding a lot more. The class hierarchy we designed for different types of munitions makes it easy to add a different weapon class using existing behavior or creating new ones. Our standard missile, for example, follows a linear path from its origin to its target. Our design separated the projectile class from the class that controls its motion. We currently implement a LineSolver class to control the standard missile. Adding more complex motion therefore simply means adding a new MotionSolver class (for simple curves, perhaps a ParabolicSolver).

Defensive Measures – In addition to new weapons, we'd like to add items like enhanced shields and other means of deflecting incoming fire.

Game Progression – The demo has a single goal, to achieve a high score. This will be completely replaced with a level based system of progression. Each level will start with the user selecting weapons from their arsenal (with a planned maximum of five) and starting the level with a limited amount of ammunition for each weapon. In between levels, users will be allowed to "shop" for new weapons and defensive measures using a currency collected based on their score on the previous level. This progression will be similar to U.N. Squadron[1] and various other games.

Mobile Launching Platforms – This ties in with the accelerometer. The concept is to have a mobile launching platform for certain weapons that can slide from one side of the screen to the other by tilting the device. Speed varies based on the angle of tilt. This would serve to break up

the standard game play by allowing the user a trade off between the weapons available (a max of 3 instead of 5), and having a mobile base and the ability to dodge incoming fire.

Bosses – The idea of having larger launching platforms that have to be destroyed in addition to the projectiles they are tossing at the player is intriguing. As an example, imagine a large bomber passing over the player's base and dishing out several types of projectiles, forcing the player to adjust for each type. In addition, the player must also pan up the camera to bring the bomber into view and destroy it. This ties in well with the planned use of the accelerometer in addition to creating a more exciting arcade experience.

Multiplayer – This would be a natural extension to the boss concept. Instead of being computer controlled, the opposing weapons platform would be player controlled, allowing players to compete head to head all over the world. A real time implementation would be difficult given the latency of the internet connection on current cell phones. A turn based system could be developed, allowing a player to design a volley of different projectiles and then launch them. The volley data would be transmitted to the other player, who would then have to fend off the volley. This could go back and forth until both players run out of ammunition (at which point a high score would determine the winner) or until one of the player's weapons platforms was destroyed.

Online Content – This feature would revolve around a website that includes profile information about all Boom players who wanted to take part (weapon platform configurations, high scores, head-to-head match replays, etc). In addition, players would have access to search features and new content (levels, weapons, etc.) when they became available.

ADC Judging Criteria

The following is a self evaluation regarding the four judging criteria laid out by Google for the Android Developer Challenge:

1. Originality – While we believe the code base we've built thus far is a solid foundation, the demo is only the tip of the iceberg. Most of the planned features that would make this game truly original on a mobile platform have yet to be implemented. This has been partially due to the learning curve associated with any new platform or SDK, but also because there is no hardware widely available to test the advances features on. We look forward to getting our hands on some hardware and revisiting this for Phase 2.
2. Effective Use of the Android Platform – Here we shine in three key areas:
    a. UI Implementation –Buttons and text fields are used sparingly for the UI so as not to clutter the screen or hinder the game play.
    b. Game Experience – We believe the decision to use the Canvas calls to render the game and using the Camera class to adjust the user's perspective enhanced the game experience without sacrificing performance.
    c. Performance – Careful consideration was given to the emulator's performance when designing Boom. The balance between the two primary threads (UI and Canvas) is crucial to keeping the performance up to par (> 15 fps in most cases). We hope that the actual hardware will be capable of handling much more.

3. Polish and Appeal – The game is both aesthetically pleasing and easy to use thanks to excellent graphic design work and Android's UI library.
4. Indispensability – While it's hard to imagine any game being indispensable (World of Warcraft player's will disagree of course), we can say that after several play testing sessions, the game is addictive. Again, the demo is only the tip of the iceberg as far as the possibilities are concerned. Our current high score stands at around 25,000 points, if anyone wants to try and beat it.

Known Issues (Bugs)

Based on our internal play testing, these are all current issues with the Boom demo:

[TBD]

References

1. U.N. Squadron (Wikipedia) - http://en.wikipedia.org/wiki/U.N._Squadron
2. Android Camera Class - http://code.google.com/android/reference/android/graphics/Camera.html
3. Android SurfaceView Class - http://code.google.com/android/reference/android/view/SurfaceView.html
4. Missile Command (Wikipedia) - http://en.wikipedia.org/wiki/Missile_Command
5. Particle Systems in Java - http://www.jhlabs.com/java/particles.html
6. Android MotionEvent Class - http://code.google.com/android/reference/android/view/MotionEvent.html

Credits

We want to thank and congratulate the Android team at Google for an awesome SDK. The entire development process was painless thanks to the tools and services you guys provided. Everything including the Eclipse plugin, the Android web page and SDK reference, the API examples, and the Google Code SVN vob worked well and made this a very enjoyable experience.

Also, if you guys have actually read all of this, we thank you for the time and consideration.

The Boom Team
Erik Bolton – UI/Rendering Code
Evan Yamada – Game Logic/Scoring Code
Jeff Budd - Graphics