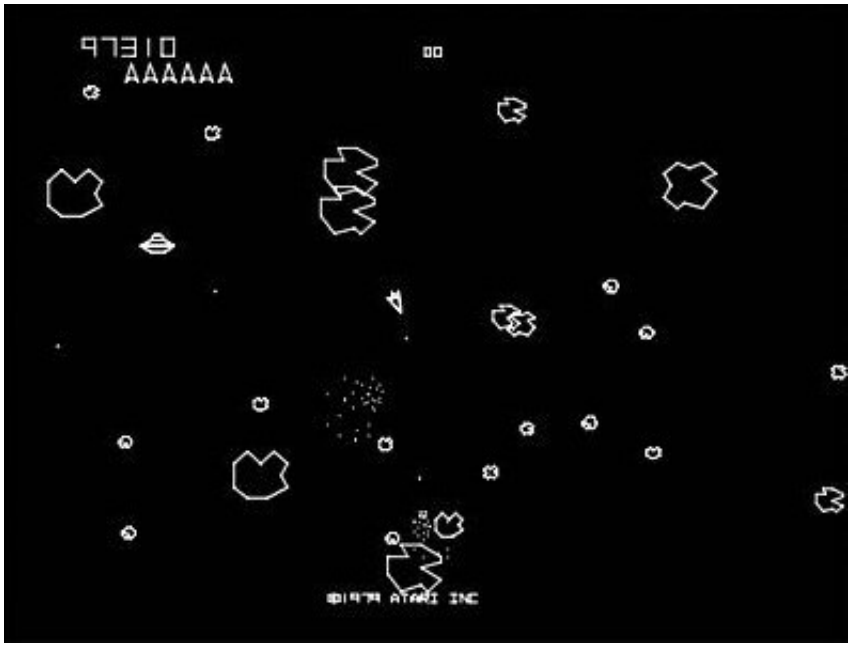
⬅ Project 2: Asteroids

Due Saturday 3/21 by 11:59:59PM (or late Sunday for -10 points)

This is a brand new project. There are probably some typos or issues that will need to be worked out, but I've tried my hardest to make sure it's all good. If you're unsure of something or you spot a mistake, please let me know!

For this project, you'll be making a sort of clone of the immensely popular **1979 arcade game *Asteroids***. Don't worry, **you aren't going to implement everything from scratch!** Instead, I've given you a very simple game engine and you'll be implementing all the interesting logic to make this game work.



Contents:

Grading Rubric

The numbers in brackets are point values. **I split up the points into very small categories** so that you can use this as a kind of checklist.

- **[10] Submission and code style**
  - **Please follow the submission instructions at the bottom of this page.**
  - **Code style is important.**
  - You should be following the calling conventions and register usage conventions.
  - You should also use multiple functions, where they make sense. (I've already given you many functions, but feel free to add more.)
- **[40] Player object** *(points below are sub-categories)*
  - **[18]** for movement:
    - **[4]** rotates with left/right keys
    - **[2]** accelerates with up key
    - **[6]** generates thrust vector from angle
    - **[2]** applies acceleration
    - **[4]** damps and applies velocity; wraps position
  - **[10]** for firing bullets:
    - **[2]** fires bullets with B key
    - **[2]** creates bullets at player's position
    - **[6]** there is a delay between bullets using the `player_fire_time` timer
  - **[12]** for taking damage, respawning, etc:
    - **[2]** *when not invulnerable,* takes damage (health is reduced)
    - **[2]** gets **temporary** invulnerability frames when damaged but not dead
    - **[4]** loses life and is disabled/disappeared when health reaches 0 (deadframes)
    - **[2]** respawns if lives are left
    - **[2]** game over if no lives left
- **[10] Bullet object**

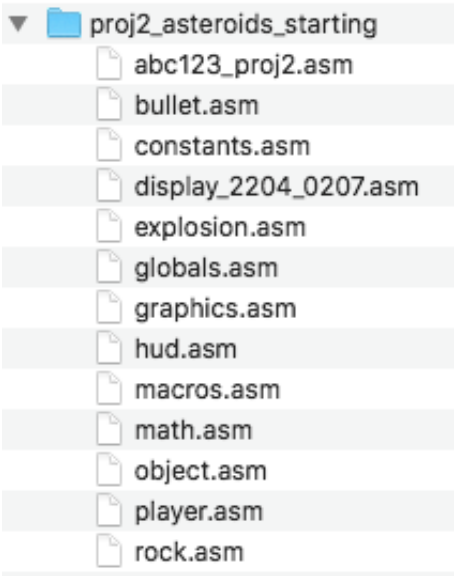
    - **[2]** `bullet_new` creates bullet object and checks for null
    - **[2]** `bullet_new` sets bullet's velocity vector and lifetime counter
    - **[2]** `bullet_update` decrements the lifetime counter and deletes it when it reaches 0
    - **[2]** `bullet_update` otherwise accumulates velocity/wraps position
    - **[2]** `bullet_draw` draws the bullet
- **[30] Rock object**
    - `rocks_init`:
        - **[2]** makes as many rocks as given by the argument
        - **[2]** positions them randomly using the formula in the instructions
    - `rock_new`:
        - **[2]** creates rock object and checks for null
        - **[2]** sets velocity vector to random direction
        - **[2]** sets speed and bounding box based on type
    - `rock_get_hit`:
        - **[2]** large rocks split into 2 medium rocks
        - **[2]** medium rocks split into 2 small rocks
        - **[2]** rock object is destroyed
    - `rock_collide_with_bullets`:
        - **[2]** properly loops over `objects` array
        - **[2]** finds `TYPE_BULLET` objects and skips others
        - **[2]** checks if the bullet's position is inside the rock
        - **[2]** destroys rock and bullet if so
    - **[2]** `rock_update` accumulates velocity/wraps position
    - **[2]** `rock_collide_l/m/s` call `rock_get_hit` and `player_damage`
    - **[2]** `rock_draw_l/m/s` draw the rocks
- **[10] Explosion object**
    - **[2]** `explosion_new` creates and initializes the object
    - **[2]** `explosion_update` updates the animation timer/frame
    - **[2]** `explosion_update` deletes the object when the animation is over

- **[2]** `explosion_draw` draws the correct frame
- **[2]** explosions are created when player or rocks are destroyed

---

Getting started

Right-click and download this ZIP file. Your browser may automatically extract it to a folder. If not, open it and extract its contents to a **new folder.**

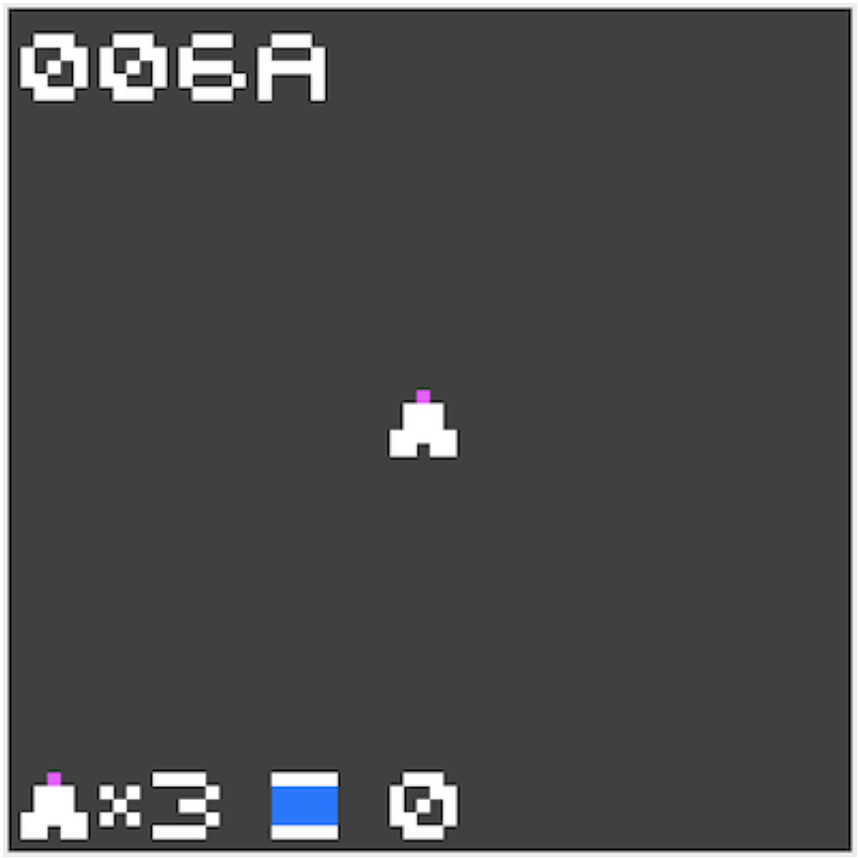


Important: Because of a MARS setting we'll be using, you **must not have any other** `.asm` **files in the same directory as these files.** See my directory on the right? That's what yours should look like. No other files in there.

Now:

1. **Rename** `abc123_proj2.asm` to your username.
2. Open that file in MARS.
3. Enable **Settings > Assemble all files in directory.**
   - *Note: you'll have to turn this off if you want to assemble/run your other labs/project that you already made.*
4. Open and connect the **Keypad and LED Display Simulator** tool.
5. **Assemble and run.**

It should look like this, with the number in the top-left increasing quickly:



What you've been given

You have been given a **very simple game engine.** A *game engine* is a library - a collection of functions - which handles a lot of the more tedious and repetitive parts of making a game, so that you can focus on the interesting parts. There are also many "stubbed-out" (blank) functions which **you will be filling in.**

An overview of the source code:

- **Files that you won't *have to* modify** (but you're allowed to, if you're adding something to them):
  - `macros.asm` - a collection of useful macros, which you `.include`.
  - `constants.asm` - all the named constants, which you `.include`.
  - `globals.asm` - all the global variables.

    - `display_2204_0207.asm` - the Keypad and LED Display driver, with many drawing functions.
    - `math.asm` - several mathematical/trigonometric functions.
    - `object.asm` - many functions for allocating/destroying/updating game objects.
- **Files that you *might* modify:**
    - `graphics.asm` - the images for all the game objects, as arrays.
    - `hud.asm` - drawing the "heads-up display" - the symbols and numbers onscreen.
        - you can change this to display debugging information - very useful!
- **Files that you *will* modify:**
    - `abc123_proj2.asm` - the main state machine function which drives the entire program.
        - You'll just put your name at the top and uncomment one part later.
    - `player.asm` - the spaceship that the player controls.
    - `bullet.asm` - the bullets that the player fires.
    - `rock.asm` - the rocks that the player destroys.
    - `explosion.asm` - explosion effects for when rocks or the player are destroyed.

Click here for the reference manual for this mini-engine.

Your task

You will **implement four game objects:** the player; the player's bullets; the rocks; and explosion effects.

Click here for instructions on implementing these objects.

Submission

Be sure to review the grading rubric before submitting.

You will submit a ZIP file containing:

- Your `abc123_proj2.asm` file (but renamed with your username).
  - *Put your name and username at the top of the file in comments.*
  - **Also put any important notes to the TA at the top of this file in comments.**
- **All** the other `.asm` files I gave you.
- Any other `.asm` files you added.

To make a ZIP file:

1. In your file browser, select all the files you want to add to the ZIP file (the files listed above).
2. **Right click on the files,** and…
   - **Windows:** do **Send To > Compressed (zipped) folder**. Then you can name it.
   - **macOS:** do **Compress *n* items**. Then you can rename the `Archive.zip` file.
   - **Linux:** I'm sure you already know.

Then, once you've made the ZIP file, **make sure to name it correctly.** My

username is `jfb42`, so:

- ✅ `jfb42_proj2.zip` - the one and only acceptable filename.
- ❌ `jfb42_proj2` - no extension
- ❌ `JFB42_proj2.zip` - uppercase is bad
- ❌ `jfb_proj2.zip` - incomplete username
- ❌ `proj2.zip` - no username
- ❌ `jfb42_project2.zip` - it's `proj2`, not `project2`
- ❌ `jfb42_proj02.zip` - it's `proj2`, not `proj02`
- ❌ literally anything other than the first thing on this list

Submit here. Drag your asm file into your browser to upload. **If you can see your file in the folder, you uploaded it correctly!**

You can also re-upload to resubmit. It will overwrite your old submission (but we can still access the old one through Box).

Just so we're clear, **if you submit on Saturday, and then resubmit on Sunday, you will get the late penalty.** But hey, if you submit a 60%-worthy project on Saturday and then a 90%-worthy one on Sunday, minus the 10% penalty, that's still an 80%!