# Galaxian Game Project

**Phase 2**

## Overview

This document provides a description of the second phase of the project. The default project is the development of a simplified version of the classic arcade game Galaxian. The project will be described, developed, and delivered in four phases. Each phase will add additional functionality to the previous phase.

Refer to the document **Galaxian Project Goals.docx** for descriptions on how the project is to be graded and submitted.

## Phase 2 Requirements

This section provides the additional requirements (appended on Phase 1 Requirements) that are to be submitted for grading on the due date.

* The ship fires a rocket sprite from its location to the top of the window when the space bar is pressed.
* Only one rocket at a time can be fired (can be in flight) at a time.
* When a rocket leaves the top of the window, the firing of new rocket is enabled.
* A collision between a grid monster and the rocket will remove the monster and rocket.
* A collision immediately enables rocket firing.
* When all of the grid monsters have been removed, reinitialize the monsters in the grid.



## Before You Start

Your team should have completed the first phase of the project. This second phase project will build on the first.

Make a copy of your first phase sketchbook into a new project-two sketchbook. Do not overwrite / lose your first phase delivery.

## Sprites

A complete understanding of the Sprites library will be needed to complete this project. Be sure to read the slides “Sprites Materials”. You can also download the source code from the library site which provides 4 example applications.

The Sprite library API can be found at <http://www.lagers.org.uk/s4p/ref/index.html>

## Build the Rocket

The function buildRocket() must be implemented and is not being provided. I suggest using buildMonster() as an example and change the sprite to use the correct image / .png file. Note unlike a monster sprite, the rocket sprite should initially be invisible as it’s not seen until the player fires with the space bar. This is accomplished by setting the rocket sprite dead using setDead(true).

## Launching Rockets

The function fireRocket() must be called. The function call is found in the function checkKeys() but is commented out. Note that the function is called when the spacebar has is pressed.

Note: The following code contains errors for you to fix.

double upRadians = 4.71238898;

void fireRocket()

{

if (rocket.isDead() && !ship.isDead()) {

rocket.setPos(ship.getPos());

rocket.setSpeed(rocketSpeed, upRadans);

rocket.setDead(false);

}

}

The requirements call for only one active rocket i.e. if the user launches a rocket (presses space bar) and a rocket is currently flying, the space bar must be ignored. I suggest using the isDead() property of the rocket sprite. The rocket sprite is alive (dead = false) when it is flying. So the rocket sprite’s isDead() property can be used to ignore spacebar presses.

When fired, the rocket must travel from the ship’s current X position straight up towards the top of the window. The implementation of fire shown above includes:

1. When fired, set the rocketSprite’s position to the ship’s current position i.e. with setPos() and getPos() i.e. rocket.setPos(ship.getPos()).
2. Set the rocket sprite’s speed and direction (setSpeed()) to cause the rocket to fly towards the window top without manually updating its position on each frame. Adjust the speed to find a setting that is both comfortable (not too fast / not too slow).
3. Use setDead(false) to start the rocket flying.

## Detecting Rocket - Monster Miss

While the rocket is flying, there are two possible outcomes:

1. The rocket collides with a monster. This is covered in the next section on detecting collisions.
2. The rocket misses all monsters and passes through the top of the window.

The Miss Monster case can be detected in the pre() function using the following code segment. Notice the combination of !rocket.isDead() (i.e. rocket is not dead) and !isOnScreem()(i.e. not on screen). (Note the method is misspelled in the S4P library).

Note: The following code contains errors for you to fix.

// If rocket flies off screen

if (!rockt.isDead() && !rockt.isOnScreem()) {

stopRocket();

}

A rocket is stopped in the function stopRocket() by setting the rocket sprite’ speed to zero and it to dead setDead(true). With the rocket marked dead, it is eligible for re-firing with the next space bar press. You must add stopRocket() to your program.

void stopRocket()

{

rocket.setSpeed(0, upRadians);

rocket.setDead(true);

}

## Detecting Rocket - Monster Collisions

The Sprites library provides built-in support for detecting the collision between two sprites i.e. whether two sprites are overlapping. There are several collision methods provided by the Sprite class and each of has a different method of identifying a collision. Any of these methods should work for your game.

Collisions between sprites should be checked for each frame. This can be accomplished in the empty processCollisions() function that was provided in the phase one implementation. Implement the following collision detection between the rocket and any of the grid monsters in the processCollisions() function.

Note: The following code must be placed in nested for loops that will compare each of the N\*M monsters. See buildMonsterGrid() for an example.

// Between Grid Monster and Rocket

Sprite monster = monsters[idx][idy];

if (!**monster**.isDead() && !rocket.isDead() && **rocket**.bb\_collision(**monster**)) {

monster.setDead(true);

rocket.setDead(true);

}

The collision detection process should include:

1. Only check for a collision with the rocket sprite if the rocket is NOT dead (!isDead() ) i.e. when the rocket is flying. Do not identify collisions when the rocket sprite is dead.
2. Iterate through each of the monsters (nested for loops are one way) and test whether the rocket sprite collides with one of the monsters. The first monster that collides with the rocket should be marked dead and the rocket also marked dead.

## Detecting Empty Grid and Replacing Monster Grid

Implementation Tip: Try reducing the number of grid monsters to something like 2x2 to make it easier to clear the grid and test grid replacement. (See variables monsterCols & monsterRows)

The requirements for the game state that once all of the grid monsters have either been destroyed or left the screen as flying monsters, the grid should be repopulated with NxM monsters and the game continued. Note that we mark monsters that are ‘shot’ as dead (setDead( true) ).

One possible method of determining whether to repopulate the grid is to check in pre() whether all the monsters are dead. This can be accomplished in a nested for loop that returns the first non-dead monster that is encountered or null if none is found. The following function pickNonDeadMonster() has been provided that provides this functionality.

// Pick the first monster on the grid that is not dead.

// Return null if they are all dead.

Sprite pickNonDeadMonster()

{

for (int idy = monsterRows; idy > 0; idy-- ) {

for (int idx = monsterCols; idx > 0; idx-- ) {

Sprite monster = monsters[idx-1][idy-1];

if (!monster.isDead()) {

return monster;

}

}

}

return null;

}

Note that pickNonDeadMonster() will be useful in phase 3 when a non-dead grid monster must be identified to become a flying monster.

The function pickNonDeadMonster() should be called in an if statement in the pre() function. If pickNonDeadMonster() returns null, you need to call resetMonsters() to reset the grid. For example:

if(pickNonDeadMonster() == null) {

resetMonsters();

}

Resetting the monster grid is accomplished by the resetMonsters() function. Resetting the monster grid is accomplished in a nested loop that places each monster in the correct grid location and sets dead = false.