# Galaxian Game Project

**Phase 3**

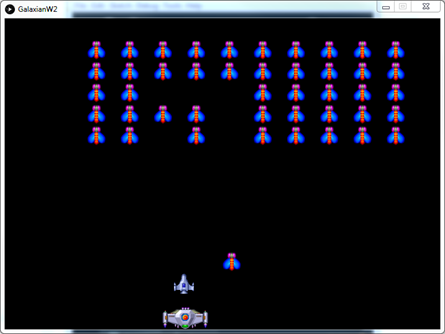
## Overview

This document provides a description of the third 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 3 Requirements

* Periodically one monster (i.e. the Falling Monster), drops from its position in the grid and moves towards the bottom of the window. How the FM is chosen is implementation dependent.
* To make the Falling Monster more difficult to hit with a rocket, have it randomly veer left and right as it travels towards the bottom.
* A Falling Monster that hits either the left or right side of the window will bounce back in the opposite direction of travel.
* A Monster that reaches and leaves the bottom of the window will leave the game i.e. do not place it back into the grid. Assignment 2 requirement about rebuilding the grid when all monsters leave still apply.
* Detect when a rocket collides with the Falling Monster and to remove both monster and rocket from window. Re-enable rocket fire.
* Detect when a Falling Monster collides with the ship and remove both monster and ship from the window.
* A collision between a flying monster and the ship ends the game and no more rockets can be fired.



## Before You Start

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

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

## Select Falling Monster

The first task is to see if a monster is already flying and to select and launch a new flying monster if not. The following code can be implemented in the pre() function.

Note: This code segment contains four missing variables declarations to be added.

// Add new flying monster if none exist

if (flyingMonster == null) {

flyingMonster = pickNonDeadMonster();

if (flyingMonster == null) {

resetMonsters();

flyingMonster = pickNonDeadMonster();

}

double direction = (int(random(2)) == 1) ? fmRightAngle : fmLeftAngle;

flyingMonster.setSpeed(fmSpeed, direction);

// Domain keeps the moving sprite withing specific screen area

flyingMonster.setDomain(0, 0, width, height+100, Sprite.REBOUND);

}

Note: The variables fmSpeed and fmRightAngle are described in the following section.

Note: The function pickNonDeadMonster() was provided in the second phase.

Note: We are using S4P’s ability to set speed and duration, along with a domain to move and constrain the flight of the FM. This is discussed further in the next section.

## Falling Monster Flight

The code highlighted in red above is responsible for the flight of the flying monster from its position in the grid to the bottom of the game window. This flight is implemented using S4P’s ability to set the speed and direction of a sprite and for the position of the sprite to be automatically updated from one frame to the next (S4P.updateSprites()).

In terms of the FM flight, the code above builds a domain equal to the size of the game window but extends past the window’s bottom by 100 pixels (height+100). This allows the FM to drop off the bottom of the screen where it can be marked dead.

The speed and direction of the FM are provided by the following variables. The angles describe the angle of descent when the monster is traveling from left to right (fmRightAngle) and from right to left (fmLefttAngle). The speed of this movement is specified by fmSpeed.

double fmRightAngle = 0.3490; // 20 degrees

double fmLeftAngle = 2.79253; // 160 degrees

double fmSpeed = 150;

The position of the FM must be checked to determine when it both misses the ship and any rockets, and drops off the bottom of the screen. This is logic that can be implemented in the pre() function.

// If flying monster is off screen

if(flyingMonster != null && !flyingMonster.isOnScreem()) {

flyingMonster.setDead(true);

flyingMonster = null;

}

This code checks if the FM is off the screen (only the bottom is possible) and marks it dead.

The code also sets the flyingMonster variable to null as a flag by other functions in the game to signal that the FM is no longer active.

## Random Flying Monster Movement

The Flying Monster is to randomly switch its direction of flight. This can be accomplished with the following code placed in the moveMonster() function provided in the first phase sketchbook.

Note: This code segment contains one missing variable declaration to be added.

// Move Flying Monster

if (flyingMonster != null) {

if (int(random(difficultity)) == 1) {

// Change FM Speed

double newSpeed = flyingMonster.getSpeed() + random(-40,40);

flyingMonster.setSpeed(newSpeed);

// Reverse FM direction.

if(flyingMonster.getDirection() == fmRightAngle)

flyingMonster.setDirection(fmLeftAngle);

else

flyingMonster.setDirection(fmRightAngle);

}

}

Processing provides a function random(N) which will return a random number between 1 and N. If the direction change occurs for a return value of 1, there is a 1 / N chance of a direction change occurring.

Likewise, the Flying Monster can also randomly change its speed to make it more difficult to hit with a rocket. Processing provides a different random number function **random(N,M)**, which returns a random number between N and M. As shown above, N->M can be in a negative to positive range causing the FM flight to slow or speed up against the given interval.

Also consider the following change. You would like the flying monster to move independently of the grid monsters. This can be accomplished by updating the following code in moveMonsters() and adding the text highlighted in red to the if statement in the two nested for loops:

if (!monster.isDead() && monster != flyingMonster) {

## Falling Monster Collision with Rocket

When the Falling Monster collides with the rocket, both the flying monster and rocket need to be removed. The following code is a test between the rocket and FM can be implemented in the function processCollisions() provided the version 2.

// Between Flying Monster and Rocket

if (flyingMonster != null && !flyingMonster.isDead()

&& !rocket.isDead() && rocket.bb\_collision(flyingMonster)) {

flyingMonster.setDead(true);

rocket.setDead(true);

flyingMonster = null;

}

## Falling Monster Collision with Ship

When the Falling Monster collides with the ship sprite, the game is considered to be over. The following code is a test between the ship and FM can be implemented in the function processCollisions() provided the version 2.

Note: The following code requires two new functions and a new Boolean variable gameOver which is to be initialized to false.

// Between Flying Monster and Ship

if (flyingMonster != null && !ship.isDead()

&& flyingMonster.bb\_collision(ship)) {

explodeShip();

monsterHit(flyingMonster);

flyingMonster = null;

gameOver = true;

}

Additional logic is to be added to processCollision() to detect the collision between the FM and the ship sprites.

The function explodeShip() will be used to display an explosion animation in the final phase of the game. In this version it should simply set the ship sprite to dead so that it will not be displayed (setDead(true)).

The function monsterHit(Sprite fm) should currently only set the monster sprite to dead. The function could play a sound or animation in phase 4.

The boolean variable gameOver is initialized to true. This flag is used in phase 4.