```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="UTF-8">
	<title>Create Project</title>
	<link rel="stylesheet" href="main.css">
</head>
<body>
	<script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.16/p5.min.js"></script>
	<script src="classes/sprite.js"></script>
	<script src="classes/ui.js"></script>
	<script src="classes/enemy.js"></script>
	<script src="enemies/standard.js"></script>
	<script src="enemies/zigzag.js"></script>
	<script src="enemies/strong.js"></script>
	<script src="enemies/homing.js"></script>
	<script src="enemies/puff.js"></script>
	<script src ="enemies/big.js"></script>
	<script src="controllers/utilities.js"></script>
	<script src="objects/bullet.js"></script>
	<script src="objects/ship.js"></script>
	<script src="objects/powerups.js"></script>
	<script src="collisions/collisions.js"></script>
	<script src="controllers/waves.js"></script>
	<script src="controllers/health.js"></script>
	<script src="controllers/power.js"></script>
	<script src="game/text.js"></script>
	<script src="game/game.js"></script>
</body>
</html>

function Sprite(x, y, r, vector, t) {
	this.x = x;
	this.y = y;
	this.r = r;
	this.vector = vector;
	this.t = t;
}

Sprite.prototype.control = function() {
	this.display();
}
```

```
function UI(x, y, color) {
	this.x = x;
	this.y = y;
	this.color = color;
}

UI.prototype.control = function() {
	this.display();
}
```

```
// constructor for enemies
function Enemy(health, radius, speed, color) {
	_waveController.enemyCount++;
	this.t = "enemy";
	this.health = health;
	this.maxHealth = health;
	this.r = radius;
	this.speed = speed;
	this.color = color;
	this.x = random(0,width);
	this.y = -this.r/1.5;
	this.staticBar = false;
}

// performs the task of killing off an enemy
Enemy.prototype.die = function() {
	this.health -= 10;
	if (this.health <= 0) {
	_waveController.deadEnemies++;
	_waveController.checkWave();
	_enemies.splice(_enemies.indexOf(this),1);
	}
};

// manages an enemies movement
Enemy.prototype.constrainMovement = function() {
	// if enemy moves off the screen, it returns to the opposite side
	if (this.x > (width + this.r) || this.x < -this.r) {
	this.x = width - this.x;
	}

	//if an enemy moves off the bottom of the scree, it returns to the top
```

```
        if (this.y > (height + this.r)) {
        this.y = -this.r;
        }
};

// Checks if enemy is on screen, used to determine whether checking collisions or firing bullets
is necessary
Enemy.prototype.checkLocation = function() {
        return (this.y > -this.r && this.y < (height + this.r))
};

// Updates each frame
Enemy.prototype.control = function() {
        this.constrainMovement();
        this.act();
        if (this.checkLocation())
        calcCollisions(this, _bullets);
        noStroke();
        fill(this.color);
        ellipse(this.x, this.y, this.r, this.r);
        this.showHealthBar();
};

// controls the enemies health bar
Enemy.prototype.showHealthBar = function() {
        noStroke();
        fill(color(255,50,50));
        if (!this.staticBar) {
        rect(this.x - ((this.maxHealth + 4)/4), this.y - (this.r/1.2), (this.health + 4) / 2, this.r/7.5);
        }
        else {
        if (this.y > -this.r/2)
        rect(width/2 - ((this.maxHealth + 4)/12), 20, (this.health + 4) / 6, 10);
        }

};

// aims bullet towards player
Enemy.prototype.aimBullet = function(speed) {
        var direction = createVector(_ship.x - this.x, _ship.y - this.y);
        direction.normalize();
        direction.mult(speed);
        return direction;
```

```
};

// standard enemy constructor
function StandardEnemy(y) {
        // inherits from Enemy
        Enemy.call(this, 30, 25, 4, color(200, 100, 100));
        this.y = y - this.r/2;
        this.damage = 50;
}

StandardEnemy.prototype = Object.create(Enemy.prototype);
StandardEnemy.prototype.constructor = StandardEnemy;

// unique movement
StandardEnemy.prototype.move = function() {
        this.y += this.speed
};

// frame update
StandardEnemy.prototype.act = function() {
        this.move();
};
function ZigZagEnemy(y) {

        Enemy.call(this, 10, 25, getRandomInt(3) + 1, color(200, 0, 100));
        this.y = y - this.r/2;
        this.canFire = true;
        this.dx = random(-1,1);
        this.dy = 0;
        this.amplitude = (getRandomInt(40) + 10)/10.0;
        this.damage = 10;

}


ZigZagEnemy.prototype = Object.create(Enemy.prototype);
ZigZagEnemy.prototype.constructor = ZigZagEnemy;
ZigZagEnemy.prototype.fireDelay = 600;


ZigZagEnemy.prototype.fireBullet = function() {
        var self = this;
        if (this.canFire && this.y > -this.r) {
        _bullets.push(new Bullet(this.x,this.y,8,createVector(0, 15),"enemy",5));
        this.canFire = false;
        setTimeout(function(){ self.canFire = true;}, self.fireDelay);
        }
};

ZigZagEnemy.prototype.move = function() {
        this.dx += .05;
        this.x += sin(this.dx)*this.amplitude;
        this.y += this.speed
};


ZigZagEnemy.prototype.act = function() {
        this.fireBullet();
        this.move();
};

function StrongEnemy(y) {

        Enemy.call(this, 100, 60, 1, color(200, 100, 0));
        this.y = y - this.r/2;
        this.canFire = true;
        this.damage = 10;

}


StrongEnemy.prototype = Object.create(Enemy.prototype);
StrongEnemy.prototype.constructor = StrongEnemy;
StrongEnemy.prototype.fireDelay = 1000;


StrongEnemy.prototype.fireBullet = function() {
        var self = this;
        if (this.canFire && this.y > -this.r) {
        _bullets.push(new Bullet(this.x,this.y,15,this.aimBullet(5),"enemy",20));
        this.canFire = false;
        setTimeout(function(){ self.canFire = true;}, self.fireDelay);
        }
};
```

```
StrongEnemy.prototype.move = function() {
        if (this.y < height / 3)
        this.y += this.speed
};


StrongEnemy.prototype.act = function() {
        this.fireBullet();
        this.move();
};

function HomingEnemy(y) {

        Enemy.call(this, 10, 15, 3, color(200));

        this.vector;
        this.damage = 25;
        this.y = y - this.r/2;

}


HomingEnemy.prototype = Object.create(Enemy.prototype);
HomingEnemy.prototype.constructor = HomingEnemy;

HomingEnemy.prototype.move = function() {
        this.vector = createVector(_ship.x - this.x, _ship.y - this.y);
        this.vector.normalize();
        this.vector.mult(this.speed);
        this.x += this.vector.x;
        this.y += this.vector.y;
};


HomingEnemy.prototype.act = function() {
        this.move();
};

function PuffEnemy(y) {

        Enemy.call(this, 100, 50, 2, color(200, 0, 200));

        this.x = 0;
        this.y = y - this.r/2;
        this.canFire = true;
        this.damage = 10;
        this.rotate = false;

        this.dx = 0;
        this.amplitude = 5;
}


PuffEnemy.prototype = Object.create(Enemy.prototype);
PuffEnemy.prototype.constructor = PuffEnemy;
PuffEnemy.prototype.fireDelay = 500;


PuffEnemy.prototype.fireBullet = function() {
        var self = this;
        if (this.canFire && this.y > -this.r) {
        this.rotate = !this.rotate;
        if (this.rotate) {
        _bullets.push(new Bullet(this.x,this.y,8,createVector(8, 8),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(-8, 8),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(8, -8),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(-8, -8),"enemy",5));
        } else {
        _bullets.push(new Bullet(this.x,this.y,8,createVector(0, 8),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(8, 0),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(0, -8),"enemy",5));
        _bullets.push(new Bullet(this.x,this.y,8,createVector(-8, 0),"enemy",5));
        }
        this.canFire = false;
        setTimeout(function(){ self.canFire = true;}, self.fireDelay);
        }
};

PuffEnemy.prototype.move = function() {
        this.dx += .01;
        this.x += sin(this.dx)*this.amplitude;
        this.y += this.speed;
};


PuffEnemy.prototype.act = function() {
        this.fireBullet();
        this.move();
};

function BigEnemy(y) {

        Enemy.call(this, 800, 600, 0.5, color(140, 0, 0));
        this.x = width/2;
        this.y = y - this.r/2;
        this.damage = 100;
        this.staticBar = true;

}


BigEnemy.prototype = Object.create(Enemy.prototype);
BigEnemy.prototype.constructor = BigEnemy;


BigEnemy.prototype.move = function() {
//      if (this.y < (height/2))
        this.y += this.speed;
};


BigEnemy.prototype.act = function() {
        this.move();
};

function getRandomInt(max) {
  return Math.floor(Math.random() * Math.floor(max));
}

function Bullet(x,y,r, vector,t, d) {

        Sprite.call(this, x, y, r, 0, t);

        this.vector = vector;
        this.isEnemy = false;
        this.bulletFill = color(200,200,100);
        this.damage = d;
        if (this.t == "enemy") {
        this.bulletFill = color(200,100,100);
        this.isEnemy = true;
        }

}

Bullet.prototype = Object.create(Sprite.prototype);
Bullet.prototype.constructor = Bullet;

Bullet.prototype.move = function() {
        calcCollisions(this, _bullets);
        this.x += this.vector.x;
        this.y += this.vector.y;
}

Bullet.prototype.checkBoundaries = function() {
        return (this.y > 0 && this.y < height && this.x > -this.r && this.x < (width + this.r));
};

Bullet.prototype.display = function() {
        if (!this.isEnemy)
        powerUpCollisions(_powerUpController, this);
        if (this.checkBoundaries()) {
        this.move();
        noStroke();
        fill(this.bulletFill);
        ellipse(this.x, this.y, this.r, this.r);
        } else {
        _bullets.splice(_bullets.indexOf(this), 1);
        }

}

function Ship(x, y, r, vector, t) {

        Sprite.call(this, x, y, r, vector, t);

        this.powerActive = false;
        this.canDie = true;
        this.canFire = true;
        this.triBullet = false;
        this.color = color(100,200,100);
```

```
		this.fireDelay = 200;
		this.bulletSpeed = -30;
		this.bulletDamage = 5;
		this.health = 100;
		this.speed = 4;
		this.vector = createVector(0,0);
		this.healthBoost = false;

	}

	Ship.prototype = Object.create(Sprite.prototype);
	Ship.prototype.constructor = Ship;

	Ship.prototype.fireBullet = function() {
		var self = this;
		if (this.canFire) {
		if (!this.triBullet) {
			_bullets.push(new Bullet(this.x,this.y - this.r/2,8,createVector(0,
	this.bulletSpeed),"player",this.bulletDamage));
			this.canFire = false;
			setTimeout(function(){ self.canFire = true;}, self.fireDelay);
		} else if (this.triBullet) {
			_bullets.push(new Bullet(this.x,this.y - this.r/2,8,createVector(0,
	this.bulletSpeed),"player",this.bulletDamage));
			_bullets.push(new Bullet(this.x+this.r/2,this.y - this.r/2,8,createVector(0,
	this.bulletSpeed),"player",this.bulletDamage));
			_bullets.push(new Bullet(this.x-this.r/2,this.y - this.r/2,8,createVector(0,
	this.bulletSpeed),"player",this.bulletDamage));
			this.canFire = false;
			setTimeout(function(){ self.canFire = true;}, self.fireDelay);
		}
		}
	};

	Ship.prototype.die = function(damage) {
		if (this.canDie) {
		if (this.health > 0) {
			this.health -= damage ? damage : 5;
		} else {
			this.health = 100;
	//			console.log("enemies: " + _enemies.length);
	//			console.log("bullets: " + _bullets.length);
	//			startGame();
```

```
                    _gameState = 1;
          }
          }
};

Ship.prototype.getKeys = function() {
          if ((keyIsDown(65) || keyIsDown(LEFT_ARROW))  && (this.x > 0)) {
          this.horizontal = -1;
          } else if ((keyIsDown(68) || keyIsDown(RIGHT_ARROW)) && (this.x < width)) {
          this.horizontal = 1;
          } else {
          this.horizontal = 0;
          }

          if ((keyIsDown(87) || keyIsDown(UP_ARROW)) && (this.y > 0)) {
          this.vertical = 1;
          } else if ((keyIsDown(83) || keyIsDown(DOWN_ARROW)) && (this.y < height)) {
          this.vertical = -1;
          } else {
          this.vertical = 0;
          }

          if (keyIsDown(32)) {
          this.fireBullet();
          }
          this.vector.set(this.horizontal, this.vertical);
          this.vector.normalize();
          this.vector.mult(this.speed);
}

Ship.prototype.choosePowerUp = function(num) {
          switch (num) {
          case 0:
          this.color = color(10, 200, 200);
          this.r *= 0.5;
          break;
          case 1:
          this.color = color(10, 100, 200);
          this.fireDelay /= 3;
          break;
          case 2:
          this.canFire = false;
          this.color = color(50, 250, 50);
```

```
        this.r *= 3;
        this.canDie = false;
        break;
        case 3:
        this.color = color(100, 200, 200);
        this.triBullet = true;
        this.fireDelay /= 2;
        break;
        case 4:
        this.color = color(50,250,50);
        this.healthBoost = true;
        break;
        default:
        break;
        }
};


Ship.prototype.activatePowerup = function() {
        this.powerActive = true;
        this.choosePowerUp(getRandomInt(5));
        _powerBar.startAnimation();
};

Ship.prototype.deactivatePowerup = function() {
        this.powerActive = false;
        this.color = color(100, 200, 100);
        this.r = 30;
        this.fireDelay = 200;
        this.canDie = true;
        this.triBullet = false;
        this.healthBoost = false;
        this.canFire = true;
};

Ship.prototype.animatePowerUp = function() {
        if(this.healthBoost && this.health < 100) {
        this.health += 0.5;
        }
};


Ship.prototype.move = function() {
        this.x += this.vector.x;
        this.y -= this.vector.y;
};

Ship.prototype.display = function() {
        this.move();
        calcCollisions(this, _bullets);
        calcCollisions(this, _enemies);
        powerUpCollisions(_powerUpController, this);
        this.animatePowerUp();
        this.getKeys();
        noStroke();
        fill(this.color);
        ellipse(this.x, this.y, this.r, this.r);
};

function PowerUp() {

        Sprite.call(this,width/2,height/2, 25);

        this.canDisplay = false;
        this.powerActive = false;
        this.dx = 0;
        this.dy = 0;
        this.amplitude = 10;

}

PowerUp.prototype = Object.create(Sprite.prototype);
PowerUp.prototype.constructor = PowerUp;

PowerUp.prototype.startTimer = function(time) {
        var self = this;
        setTimeout(function(){
        self.x = random(0,width);
        self.y = random(0,height);
        self.canDisplay = true;
        self.powerActive = false;
        }, time);
};

PowerUp.prototype.activate = function() {
        var self = this;
```

```
		self.powerActive = true;
		self.canDisplay = false;
		setTimeout(function(){
		_ship.deactivatePowerup();
		self.powerActive = false;
		self.startTimer(random(2000,4000));
		}, 5000);
	};

	PowerUp.prototype.constrainMovement = function() {
		if (this.x > (width + this.r) || this.x < -this.r) {
		this.x = width - this.x;
		}
		if (this.y > (height + this.r) || this.y < -this.r) {
		this.y = height - this.y;
		}
	};

	PowerUp.prototype.move = function() {
		this.dx += random(-10,10)/50;
		this.dy += random(-10,10)/50;
		this.x += sin(this.dx)*this.amplitude;
		this.y += sin(this.dy)*this.amplitude;
	};

	PowerUp.prototype.display = function() {
		if (this.canDisplay && !this.powerActive) {
		noStroke();
		fill(100,100,200);
		this.move();
		this.constrainMovement();
		ellipse(this.x, this.y, this.r, this.r);
		}
	};

	function calcCollisions(collider,objects) {
		for (var i = objects.length-1; i > -1; i--) {
		if (dist(collider.x,collider.y,objects[i].x,objects[i].y) < (collider.r/2 + objects[i].r/2) &&
	(collider.t != objects[i].t)) {
		if (collider.t == "player") {
			if (_enemies.indexOf(objects[i]) != -1) {
			_ship.die(objects[i].damage);
			objects[i].die();
```

```
                    _score++;
                    } else if (_bullets.indexOf(objects[i]) != -1) {
                    _ship.die(objects[i].damage);
                    objects.splice(objects.indexOf(collider),1);
                    }
          } else if (collider.t == "enemy") {
                    if (_enemies.indexOf(collider) != -1) {
                    _enemies[_enemies.indexOf(collider)].die();
                    _score++;
                    } else {
                    objects.splice(objects.indexOf(collider),1);
                    }
                    objects.splice(i,1);
          }
          }
          }
}


function powerUpCollisions(powerUpObject, checkObject) {
          if (dist(powerUpObject.x, powerUpObject.y, checkObject.x, checkObject.y) <
(checkObject.r/2 + powerUpObject.r/2) && powerUpObject.canDisplay == true) {
          powerUpObject.activate();
          _ship.activatePowerup();
          }
}
```

```
// Manages waves
function WaveController() {

          var self = this;
          // The verticle distance between enemies when spawned
          self.offset = -200;
          // Initial number of enemies
          self.enemyCount = 0;
          // Number of Dead Enemies
          self.deadEnemies = 0;

          // Generates random integer between min and max parameters
          var generateRandomNumber = function(min, max) {
          return Math.floor(Math.random() * (max - min + 1)) + min;
          };
```

```
// Returns a scaled list by applying the scale values to corresponding list values
self.createEnemyList = function(list, scale) {
var scaledList = [];
//Adds a number of enemy times equal to its corresponding scale value
for (var i = 0; i < scale.length; i++) {
for (var j = 0; j < scale[i]; j++) {
        scaledList.push(list[i]);
}
}
return scaledList;
};

// List of Enemy types
self.enemyList = [0, 1, 2, 3, 4, 5];
// List of corresponding percentages/scale values
self.listScale = [20, 20, 20, 20, 15, 5];
// Generates scaled list
self.scaledEnemyList = self.createEnemyList(self.enemyList, self.listScale);

// Generates new wave of enemies
self.createWave = function() {
_waveNumber++;
for (var i = 0; i < _waveNumber; i++) {
var randomNumber = generateRandomNumber(0, self.scaledEnemyList.length);
self.chooseEnemy(self.scaledEnemyList[randomNumber], self.offset * i);
}
};

// Called when an Enemy dies, checks if all the enemies in a wave are dead
// If so, it creates a new wave
self.checkWave = function() {
if (self.deadEnemies == self.enemyCount) {
self.deadEnemies = 0;
self.enemyCount = 0;
self.createWave();
}
};

// Spawns enemies based on the type of enemy
self.chooseEnemy = function(type, y) {
switch(type) {
case 0:
        _enemies.push(new StandardEnemy(y));
                break;
        case 1:
                _enemies.push(new ZigZagEnemy(y));
                break;
        case 2:
                _enemies.push(new StrongEnemy(y));
                break;
        case 3:
                _enemies.push(new HomingEnemy(y));
                break;
        case 4:
                _enemies.push(new PuffEnemy(y));
                break;
        case 5:
                _enemies.push(new BigEnemy(y));
                break;
        default:
                break;
        }
        };

}
```

```
function HealthBar(x, y, color) {

        UI.call(this, x, y, color);

}

HealthBar.prototype = Object.create(UI.prototype);
HealthBar.prototype.constructor = HealthBar;

HealthBar.prototype.display = function() {
        if (!_ship.canDie) {
        this.color = color(50, 250, 50);
        } else {
        this.color = color(255,50,50);
        }
        noStroke();
        fill(this.color);
        rect(this.x, this.y, 15, _ship.health + 5);
};
```

```
function PowerBar(x, y, color) {

        UI.call(this, x, y, color)
        this.length = 0;
        this.shouldAnimate = false;
        this.timeValue = 5000;
}

PowerBar.prototype = Object.create(UI.prototype);
PowerBar.prototype.constructor = PowerBar;
PowerBar.prototype.stepValue = 50;
PowerBar.prototype.width = 15;

PowerBar.prototype.startAnimation = function() {
        this.length = 105;
        this.shouldAnimate = true;
};

PowerBar.prototype.display = function() {
        noStroke();
        fill(this.color);
        if (this.shouldAnimate)
        this.length -= .35;
        if (this.length < 0) {
        this.length = 0;
        this.shouldAnimate = false;
        }
        rect(this.x, this.y, this.width, this.length);
};

function ScoreText(x, y, color) {
        UI.call(this, x, y, color);
}

ScoreText.prototype = Object.create(UI.prototype);
ScoreText.prototype.constructor = ScoreText;
ScoreText.prototype.size = 18;

ScoreText.prototype.display = function() {
        textSize(this.size);
        fill(this.color);
        textAlign(LEFT,CENTER);
```

```
	text("Score: " + _score, this.x, this.y);
};


function WaveText(x, y, color) {

	UI.call(this, x, y, color);

}

WaveText.prototype = Object.create(UI.prototype);
WaveText.prototype.constructor = WaveText;
WaveText.prototype.size = 18;

WaveText.prototype.display = function() {
	textSize(this.size);
	fill(this.color);
	textAlign(LEFT,CENTER);
	text("Wave: " + _waveNumber, this.x, this.y);
};

function EnemyText(x, y, color) {

	UI.call(this, x, y, color);

}

EnemyText.prototype = Object.create(UI.prototype);
EnemyText.prototype.constructor = EnemyText;
EnemyText.prototype.size = 18;

EnemyText.prototype.display = function() {
	textSize(this.size);
	fill(this.color);
	textAlign(LEFT,CENTER);
	text("Enemies: " + (_waveController.enemyCount - _waveController.deadEnemies),
this.x, this.y);
};

function MenuText(x, y, color) {
	UI.call(this, x, y, color);
}
```

```
MenuText.prototype = Object.create(UI.prototype);
MenuText.prototype.constructor = MenuText;
MenuText.prototype.size = 18;

MenuText.prototype.display = function() {
	textSize(this.size);
	fill(this.color);
	textAlign(CENTER,CENTER);
	text("Press ENTER to begin!", this.x, this.y);
};

var _ship;
var _enemies = [];
var _score = 0;
var _bullets = [];
var _scoreText;
var _waveText;
var _enemyText;
var _powerBar;
var _healthBar;
var _waveController;
var _waveNumber = 0;
var _powerUpController;
var _gameState;
var _menuText;



function setup() {
	createCanvas(600,600);
	bg = color(40);
	createMenu();
//	startGame();
	_gameState = 1;
	frameRate(60);
}

function draw() {
	background(bg);
	switch(_gameState) {
	case 1:
	controlMenu();
	break;
```

```
		case 2:
		controlObjects();
		break;
		default:
		break;
		}
}

function controlObjects() {
		_ship.control();
		_powerUpController.control();
		for (var i = _enemies.length - 1; i > -1; i--) {
		_enemies[i].control();
		}
		for (var i = _bullets.length - 1; i > -1; i--) {
		if (_bullets[i] != null) _bullets[i].control();
		}
		_healthBar.control();
		_powerBar.control();
		_scoreText.control();
		_waveText.control();
		_enemyText.control();
}

function startGame() {
		_ship = new Ship(width/2,height/1.5,30,0,"player");
		_enemies = [];
		_score = 0;
		_bullets = [];
		_waveController = new WaveController();
		_waveNumber = 0;
		_waveController.createWave();
		_powerUpController = new PowerUp();
		_powerUpController.startTimer(2000);
		_healthBar = new HealthBar(10,10,color(255,50,50));
		_powerBar = new PowerBar(40,10,color(0,200,255));
		_scoreText = new ScoreText(70, 77, color(255));
		_waveText = new WaveText(70,30,color(240,10,10));
		_enemyText = new EnemyText(70, 52, color(240,10,10));
}

function createMenu() {
		_menuText = new MenuText(width/2,height/2,color(0,200,0));
```

```
}

function controlMenu() {
            _menuText.control();
            if (keyIsDown(13)) {
            startGame();
            _gameState = 2;
            }
}
```