

Setting the Display

At this point, we have a black screen with a yellow circle we can control with the keyboard.

Run.py

```
import pygame
from pygame.locals import *
from constants import *
from pacman import Pacman

class GameController(object):
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode(SCREENSIZE, 0, 32)
        self.background = None
        self.clock = pygame.time.Clock()

    def setBackground(self):
        self.background = pygame.surface.Surface(SCREENSIZE).convert()
        self.background.fill(BLACK)

    def startGame(self):
        self.setBackground()
        self.pacman = Pacman()

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.checkEvents()
        self.render()

    def checkEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

    def render(self):
        pygame.display.update()
        self.screen.blit(self.background, (0,0))
        self.pacman.render(self.screen)

if __name__ == "__main__":
    game = GameController()
    game.startGame()
    while True:
```

```
game.update()
```

Vector.py

```
import math
```

```
class Vector2(object):
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        self.thresh = 0.000001

    def __add__(self, other):
        return Vector2(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2(self.x - other.x, self.y - other.y)

    def __neg__(self):
        return Vector2(-self.x, -self.y)

    def __mul__(self, scalar):
        return Vector2(self.x * scalar, self.y * scalar)

    def __div__(self, scalar):
        if scalar != 0:
            return Vector2(self.x / float(scalar), self.y / float(scalar))
        return None

    def __truediv__(self, scalar):
        return self.__div__(scalar)

    def __eq__(self, other):
        if abs(self.x - other.x) < self.thresh:
            if abs(self.y - other.y) < self.thresh:
                return True
        return False

    def magnitudeSquared(self):
        return self.x**2 + self.y**2

    def magnitude(self):
        return math.sqrt(self.magnitudeSquared())

    def __str__(self):
        return "<"+str(self.x)+", "+str(self.y)+">"

    def copy(self):
        return Vector2(self.x, self.y)

    def asTuple(self):
        return self.x, self.y
```

```
def asInt(self):
    return int(self.x), int(self.y)
```

Pacman.py

```
import pygame
from pygame.locals import *
from vector import Vector2
from constants import *

class Pacman(object):
    def __init__(self):
        self.name = PACMAN
        self.position = Vector2(200, 400)
        self.directions = {STOP:Vector2(), UP:Vector2(0,-1), DOWN:Vector2(0,1), LEFT:Vector2(-1,0),
RIGHT:Vector2(1,0)}
        self.direction = STOP
        self.speed = 100
        self.radius = 10
        self.color = YELLOW

    def update(self, dt):
        self.position += self.directions[self.direction]*self.speed*dt
        direction = self.getValidKey()
        self.direction = direction

    def getValidKey(self):
        key_pressed = pygame.key.get_pressed()
        if key_pressed[K_UP]:
            return UP
        if key_pressed[K_DOWN]:
            return DOWN
        if key_pressed[K_LEFT]:
            return LEFT
        if key_pressed[K_RIGHT]:
            return RIGHT
        return STOP

    def render(self, screen):
        p = self.position.asInt()
        pygame.draw.circle(screen, self.color, p, self.radius)
```