```python
import pygame

import levels
from levels import gameLevels, flag, topGrass, sideGrassRight, sideGrassLeft, grassCornerRight,
grassCornerLeft

#initialise pygame
pygame.init()

#create screen
width, height = 1280, 720
screen = pygame.display.set_mode((width, height))
clock = pygame.time.Clock()

#game settings
fps = 120
grav = 9.8
speed = 3
jumpSpeed = 7
jumpTime = 1.35

#variables
global allowJump
jumpKeep = -1
allowJump = False

#import images
backG = pygame.image.load("assets/images/background.png").convert()
char = pygame.image.load("assets/images/char.png").convert()
lastLevel = pygame.image.load("assets/images/lastLevel.png").convert_alpha()

#add grass to levels
gameLevels = levels.addGrass(gameLevels)

#level (columns = 20, rows = 12) ([0] and [1] of each list is the range in x for the block and [2]
and [3] is the same for y)
global levelList, currentLevel, spawn
currentLevel = 0
levelList = gameLevels[currentLevel]
spawn = [levelList[0][5] * 64, levelList[0][6] * 64]

#platfrom jumpable blocks
platforms = []

#blocks with no collision
noCol = [flag, topGrass, sideGrassLeft, sideGrassRight, grassCornerRight, grassCornerLeft]

class Player():
    def __init__(self):
        self.pos = spawn
        self.skin = char
        self.fallCount = 0
        self.force = jumpSpeed
        self.falling = False
    def move(self, newPos): #moves the player as long as it isnt out of bounds
        if self.pos[0] + newPos[0] < 1216 and self.pos[0] + newPos[0] >= 0:
            self.pos = [self.pos[0] + newPos[0], self.pos[1] + newPos[1]]
    def grav(self): #calculates gravity and makes character fall unless colliding.
        gridX, gridY = ((self.pos[0] // 64)), ((self.pos[1] // 64) + 1)
        for i in range(len(levelList)):
            if (not (gridX <= levelList[i][1] and gridY == levelList[i][2])) or (not (gridX + 1 >=
            levelList[i][0] and gridY == levelList[i][2])) or levelList[i][4] in noCol:
                self.falling = True
            else:
                self.falling = False
                self.fallCount = 0
                self.pos[1] = self.pos[1] // 64 * 64
                break
        if self.falling:
            self.move([0, grav * (self.fallCount / 60)])
            self.fallCount += 1
    def draw(self): #renders player
        screen.blit(self.skin, ((self.pos[0]), (self.pos[1])))
    def jump(self): #adds upwards force to make player jump
        global allowJump
        if self.falling == False and self.force < jumpSpeed:
            allowJump = False
            self.force = 0
        self.move([0, -self.force])
        self.force = self.force - (1 / 60)

character = Player() #instance of Player class

def drawStuff(): #renders the level based on the levelList for the particular level
    for i in range((width // 64) + 1):
        for j in range((height // 64) + 1):
            for k in range(len(levelList)):
                if i >= levelList[k][0] and i <= levelList[k][1] and j >= levelList[k][2] and j <=
                levelList[k][3]:
                    screen.blit(levelList[k][4], ((i*64), (j*64)))

def drawBackG():
    for i in range((width // 64) + 1):
        for j in range((height // 64) + 1):
            screen.blit(backG, ((i*64), (j*64)))

def drawScreen(): #graphics main loop
    drawBackG()
    character.draw()
    drawStuff()
    character.grav()

def jump():
    global allowJump
    if not character.falling:
        allowJump = True
        character.force = jumpSpeed

def jumpLoop(): #checks if character can jump before allowing it
    global allowJump
    posX, posY = character.pos[0], character.pos[1]
    for i in range(len(levelList)): #collision checks
        if posX // 64 >= levelList[i][0] and posX // 64 <= levelList[i][1] and (posY // 64) ==
        levelList[i][3] and levelList[i][4] not in noCol and levelList[i][4] not in platforms:
            check1 = True
            break
        else:
            check1 = False
    posX += 64
    for i in range(len(levelList)):
        if posX // 64 >= levelList[i][0] and posX // 64 <= levelList[i][1] and (posY // 64) == 
        levelList[i][3] and levelList[i][4] not in noCol and levelList[i][4] not in platforms:
            check2 = True
            break
        else:
            check2 = False
    if check1 or check2:
        allowJump = False
        character.force = 0
    if allowJump:
        character.jump()

def moveLeft(): #left moving collision checks
    posX, posY = character.pos[0] + -speed, character.pos[1] + 63
    for i in range(len(levelList)):
        if (posX // 64 == levelList[i][1] and posY // 64 <= levelList[i][3] and posY // 64 >= 
        levelList[i][2]) and levelList[i][4] not in noCol:
            allowMove1 = False
            break
        else:
            allowMove1 = True
    posX, posY = character.pos[0] + -speed, character.pos[1]
    for i in range(len(levelList)):
        if (posX // 64 == levelList[i][1] and posY // 64 <= levelList[i][3] and posY // 64 >= 
        levelList[i][2]) and levelList[i][4] not in noCol:
            allowMove2 = False
            break
        else:
            allowMove2 = True
    if allowMove1 and allowMove2:
        character.move([-speed, 0])

def moveRight(): #right moving collision checks
    posX, posY = character.pos[0] + speed + 64, character.pos[1] + 63
    for i in range(len(levelList)):
        if (posX // 64 == levelList[i][0] and posY // 64 <= levelList[i][3] and posY // 64 >= 
        levelList[i][2]) and levelList[i][4] not in noCol:
            allowMove1 = False
            break
        else:
            allowMove1 = True
    posX, posY = character.pos[0] + speed + 64, character.pos[1]
    for i in range(len(levelList)):
        if (posX // 64 == levelList[i][0] and posY // 64 <= levelList[i][3] and posY // 64 >= 
        levelList[i][2]) and levelList[i][4] not in noCol:
            allowMove2 = False
            break
        else:
            allowMove2 = True
    if allowMove1 and allowMove2:
        character.move([speed, 0])

def detectKey(): #detects user inputs
    keys = pygame.key.get_pressed()
    if keys[pygame.K_a] or keys[pygame.K_LEFT]:
        moveLeft()
    if keys[pygame.K_d] or keys[pygame.K_RIGHT]:
        moveRight()
    if keys[pygame.K_SPACE] or keys[pygame.K_UP]:
        jump()

def checkDead(): #checks if player fell
    if character.pos[1] >= 720:
        character.pos = spawn

def checkWin(): #checks if player reached the flag object in given level
    global levelList, currentLevel, spawn
    for i in range(len(levelList)):
        if levelList[i][4] == flag:
            gridX, gridY = ((character.pos[0] // 64)), ((character.pos[1] // 64))
            if (gridX == levelList[i][0] and gridY == levelList[i][2]) or (gridX + 1 == levelList[i]
            [0] and gridY == levelList[i][2]):
                try:
                    currentLevel += 1
                    levelList = gameLevels[currentLevel]
                    spawn = [levelList[0][5] * 64, levelList[0][6] * 64]
                    character.__init__()
                except:
                    screen.blit(lastLevel, (0, 0))
                break

def main(): #main game loop
    global jumpKeep
    runGame = True
    while runGame:

        #fps limit
        clock.tick(fps)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                runGame = False

        #check for death
        checkDead()

        #jump function
        jumpLoop()

        #keyboard detection
        detectKey()

        #draw items on screen
        drawScreen()

        #check for win
        checkWin()

        #refresh screen
        pygame.display.flip()

    pygame.quit()

if __name__ == "__main__":
    main()
```