```
"""
============================================================
  2D Car Racing Game — Pygame
============================================================
Controls:
  LEFT / RIGHT arrow keys  →  move player car
  ENTER / R                →  start or restart game
  ESC                      →  quit

Requirements:
  pip install pygame
============================================================
"""

import pygame
import random
import sys

# ─────────────────────────────────────────────────────────────
#  CONSTANTS  — change these to tweak the game
# ─────────────────────────────────────────────────────────────
SCREEN_W, SCREEN_H = 600, 800        # window size in pixels

# Road layout
ROAD_LEFT   = 170                    # left edge of tarmac
ROAD_RIGHT  = 430                    # right edge of tarmac
ROAD_W      = ROAD_RIGHT - ROAD_LEFT # 260 px wide

# Lane divider dashes
DASH_H      = 40                     # height of each white dash
DASH_GAP    = 30                     # gap between dashes
DASH_W      = 8                      # dash width

# Road scroll speed (pixels per frame at 60 fps)
INITIAL_ROAD_SPEED  = 5
SPEED_INCREMENT     = 0.0015         # road gets faster over time

# Player car
PLAYER_W    = 44
PLAYER_H    = 80
PLAYER_SPEED = 5                     # horizontal movement per frame

# Enemy cars
ENEMY_W     = 44
```

```
ENEMY_H     = 80
ENEMY_SPAWN_INTERVAL = 90           # frames between spawns (decreases over time)
MIN_SPAWN_INTERVAL   = 30

# Scoring
SCORE_PER_FRAME = 1                 # raw score increments every frame

# Colours
C_SKY        = (100, 140,  80)      # grass / forest background
C_ROAD       = ( 55,  55,  60)      # tarmac
C_ROAD_LINE  = (200, 200,  50)      # yellow side lines
C_DASH       = (240, 240, 240)      # white centre dashes
C_WHITE      = (255, 255, 255)
C_BLACK      = (  0,   0,   0)
C_RED        = (210,  40,  40)
C_DARK_RED   = (140,  20,  20)
C_YELLOW     = (255, 210,   0)
C_ORANGE     = (230, 110,  30)
C_BLUE       = ( 40, 100, 200)
C_DARK_BLUE  = ( 20,  55, 130)
C_GRAY       = (150, 150, 160)
C_DARK_GRAY  = ( 80,  80,  90)
C_GREEN_DARK = ( 34,  85,  34)
C_GREEN_MID  = ( 55, 120,  55)
C_GREEN_LITE = ( 80, 155,  80)
C_BROWN      = ( 90,  55,  30)
C_SCORE_BG   = (  0,   0,   0, 150) # semi-transparent

FPS = 60


# ──────────────────────────────────────────────────────────────
#  HELPER: draw a stylised car at (cx, cy)
#  cx/cy = centre-bottom of the car
# ──────────────────────────────────────────────────────────────
def draw_car(surface: pygame.Surface,
             cx: int, cy: int,
             body_col: tuple, roof_col: tuple,
             window_col: tuple,
             facing_up: bool = True) -> None:
    """
    Draw a top-down arcade car using simple rectangles and circles.
    facing_up = True  → player car (bonnet at top of sprite)
    facing_up = False → enemy car (bonnet at bottom of sprite)
```

```
    """
    w, h = PLAYER_W, PLAYER_H
    x = cx - w // 2          # left edge
    y = cy - h               # top edge

    # ── Body ──────────────────────────────────────────────
    pygame.draw.rect(surface, body_col, (x + 4, y, w - 8, h))

    # ── Roof / cabin ──────────────────────────────────────
    cabin_h = h // 3
    cabin_y = y + h // 3 if facing_up else y + h - h // 3 - cabin_h
    pygame.draw.rect(surface, roof_col, (x + 8, cabin_y, w - 16, cabin_h))

    # ── Windshield ────────────────────────────────────────
    wind_h = cabin_h // 2
    wind_y = cabin_y if facing_up else cabin_y + cabin_h - wind_h
    pygame.draw.rect(surface, window_col, (x + 10, wind_y, w - 20, wind_h))

    # ── Headlights / tail-lights ──────────────────────────
    light_col = (255, 255, 180) if facing_up else (255, 60, 60)
    tail_col  = (255,  60,  60) if facing_up else (255, 255, 180)
    # front lights
    front_y = y if facing_up else y + h - 6
    pygame.draw.rect(surface, light_col, (x + 4,      front_y, 10, 6))
    pygame.draw.rect(surface, light_col, (x + w - 14, front_y, 10, 6))
    # rear lights
    rear_y = y + h - 6 if facing_up else y
    pygame.draw.rect(surface, tail_col,  (x + 4,      rear_y, 10, 6))
    pygame.draw.rect(surface, tail_col,  (x + w - 14, rear_y, 10, 6))

    # ── Wheels (four corners) ─────────────────────────────
    wheel_w, wheel_h = 8, 18
    for wx in (x - 2, x + w - 6):
        for wy in (y + 6, y + h - wheel_h - 6):
            pygame.draw.rect(surface, C_DARK_GRAY, (wx, wy, wheel_w, wheel_h))
            pygame.draw.rect(surface, C_GRAY,      (wx + 2, wy + 4, 4, 10))


# ═══════════════════════════════════════════════════════════
#  TREE CLASS
# ═══════════════════════════════════════════════════════════
class Tree:
    """A single tree drawn as a trunk + layered circles (pine/leafy style)."""
```

```
    def __init__(self, x: int, y: int, size: int = 1) -> None:
        self.x = x          # centre x
        self.y = y          # centre-bottom y
        self.size = size    # 1 = small, 2 = medium, 3 = large

    def draw(self, surface: pygame.Surface) -> None:
        s = self.size
        trunk_w  = 6  * s
        trunk_h  = 14 * s
        trunk_x  = self.x - trunk_w // 2
        trunk_y  = self.y - trunk_h

        # Trunk
        pygame.draw.rect(surface, C_BROWN, (trunk_x, trunk_y, trunk_w, trunk_h))

        # Layered canopy (3 overlapping circles for a pine look)
        cx, cy = self.x, trunk_y
        radii = [14 * s, 12 * s, 10 * s]
        offsets = [0, -8 * s, -16 * s]
        colors  = [C_GREEN_DARK, C_GREEN_MID, C_GREEN_LITE]
        for r, oy, col in zip(radii, offsets, colors):
            pygame.draw.circle(surface, col, (cx, cy + oy), r)


# ──────────────────────────────────────────────────────────────
#  FOREST MANAGER  — handles all trees
# ──────────────────────────────────────────────────────────────
class Forest:
    """
    Manages a scrolling forest of trees on both sides of the road.
    Trees are pre-generated in columns so the forest looks natural.
    """

    def __init__(self) -> None:
        self.trees: list[Tree] = []
        self._populate()

    def _populate(self) -> None:
        """Create an initial set of trees covering the full screen height."""
        self.trees.clear()
        # Left forest: x from 0 to ROAD_LEFT-10
        # Right forest: x from ROAD_RIGHT+10 to SCREEN_W
        for region in ("left", "right"):
            # Scatter trees in a grid with random jitter
```

```
            col_step = 60
            row_step = 80
            if region == "left":
                x_start, x_end = 20, ROAD_LEFT - 15
            else:
                x_start, x_end = ROAD_RIGHT + 15, SCREEN_W - 20

            for row_y in range(-200, SCREEN_H + 100, row_step):
                for col_x in range(x_start, x_end, col_step):
                    jitter_x = random.randint(-15, 15)
                    jitter_y = random.randint(-20, 20)
                    size = random.randint(1, 3)
                    self.trees.append(Tree(col_x + jitter_x,
                                           row_y + jitter_y + 40,
                                           size))

    def update(self, speed: float) -> None:
        """Move every tree downward and recycle off-screen trees to the top."""
        for tree in self.trees:
            tree.y += speed

        # When a tree scrolls below the screen, move it back to the top
        for tree in self.trees:
            if tree.y > SCREEN_H + 60:
                tree.y -= SCREEN_H + 300        # wrap to above screen
                # Randomise x position within its region
                if tree.x < ROAD_LEFT:
                    tree.x = random.randint(20, ROAD_LEFT - 15)
                else:
                    tree.x = random.randint(ROAD_RIGHT + 15, SCREEN_W - 20)
                tree.size = random.randint(1, 3)

    def draw(self, surface: pygame.Surface) -> None:
        # Sort back-to-front so larger (closer) trees overlap smaller ones
        for tree in sorted(self.trees, key=lambda t: t.y):
            tree.draw(surface)


# ──────────────────────────────────────────────────────────────
#  ROAD CLASS  — scrolling tarmac + lane marks
# ──────────────────────────────────────────────────────────────
class Road:
    """Draws the tarmac and animates lane markings by scrolling a y-offset."""
```

```
    def __init__(self) -> None:
        self.scroll_y = 0.0   # tracks dash animation offset

    def update(self, speed: float) -> None:
        """Advance the scroll offset, wrapping at the end of one dash cycle."""
        self.scroll_y = (self.scroll_y + speed) % (DASH_H + DASH_GAP)

    def draw(self, surface: pygame.Surface) -> None:
        # ── Tarmac ──────────────────────────────────────
        pygame.draw.rect(surface, C_ROAD,
                         (ROAD_LEFT, 0, ROAD_W, SCREEN_H))

        # ── Yellow edge lines ────────────────────────
        line_w = 5
        pygame.draw.rect(surface, C_ROAD_LINE,
                         (ROAD_LEFT, 0, line_w, SCREEN_H))
        pygame.draw.rect(surface, C_ROAD_LINE,
                         (ROAD_RIGHT - line_w, 0, line_w, SCREEN_H))

        # ── Centre dashed line ───────────────────────
        centre_x = (ROAD_LEFT + ROAD_RIGHT) // 2 - DASH_W // 2
        # Start above screen so dashes scroll in smoothly
        y = -DASH_H + self.scroll_y
        while y < SCREEN_H:
            pygame.draw.rect(surface, C_DASH,
                             (centre_x, int(y), DASH_W, DASH_H))
            y += DASH_H + DASH_GAP


# ──────────────────────────────────────────────────────────
#  PLAYER CLASS
# ──────────────────────────────────────────────────────────
class Player:
    """The car the player controls — moves only horizontally."""

    def __init__(self) -> None:
        # Start at centre of road, near bottom of screen
        self.cx = (ROAD_LEFT + ROAD_RIGHT) // 2
        self.cy = SCREEN_H - 100
        # Collision rect (slightly inset for fairness)
        self.rect = pygame.Rect(0, 0, PLAYER_W - 8, PLAYER_H - 8)
        self._update_rect()

    def _update_rect(self) -> None:
```

```
        """Keep the collision rect centred on the car position."""
        self.rect.centerx = self.cx
        self.rect.bottom  = self.cy

    def handle_input(self, keys: pygame.key.ScancodeWrapper) -> None:
        """Read arrow keys and move the car horizontally."""
        if keys[pygame.K_LEFT]:
            self.cx -= PLAYER_SPEED
        if keys[pygame.K_RIGHT]:
            self.cx += PLAYER_SPEED

        # Clamp inside the road (leave room for car width)
        half_w = PLAYER_W // 2
        self.cx = max(ROAD_LEFT  + half_w, self.cx)
        self.cx = min(ROAD_RIGHT - half_w, self.cx)

        self._update_rect()

    def draw(self, surface: pygame.Surface) -> None:
        draw_car(surface, self.cx, self.cy,
                 body_col=C_BLUE, roof_col=C_DARK_BLUE,
                 window_col=(160, 210, 255),
                 facing_up=True)


# ──────────────────────────────────────────────────────────────
#  ENEMY CAR CLASS
# ──────────────────────────────────────────────────────────────
class Enemy:
    """An oncoming car that spawns at the top and moves downward."""

    # Different car colour schemes for variety
    COLOUR_SETS = [
        (C_RED,        C_DARK_RED,   (255, 140, 140)),
        (C_YELLOW,     C_ORANGE,     (255, 240, 180)),
        ((80, 180, 80),(30, 100, 30),(160, 230, 160)),
        ((180, 80,180),(100,30,100),(220, 160, 220)),
        ((200,200,200),(100,100,100),(240, 240, 240)),
    ]

    def __init__(self, road_speed: float) -> None:
        # Pick a random lane position (avoid hugging the edges)
        margin = ENEMY_W // 2 + 10
        self.cx = random.randint(ROAD_LEFT + margin, ROAD_RIGHT - margin)
```

```
        self.cy = -ENEMY_H    # spawn above the screen

        # Enemy moves at road speed + a small random bonus
        self.speed = road_speed + random.uniform(0, 2)

        colours = random.choice(self.COLOUR_SETS)
        self.body_col   = colours[0]
        self.roof_col   = colours[1]
        self.window_col = colours[2]

        self.rect = pygame.Rect(0, 0, ENEMY_W - 8, ENEMY_H - 8)
        self._update_rect()

    def _update_rect(self) -> None:
        self.rect.centerx = self.cx
        self.rect.bottom  = self.cy

    def update(self) -> None:
        """Move the enemy car downward each frame."""
        self.cy += self.speed
        self._update_rect()

    def is_off_screen(self) -> bool:
        return self.cy > SCREEN_H + ENEMY_H

    def draw(self, surface: pygame.Surface) -> None:
        draw_car(surface, self.cx, self.cy,
                 body_col=self.body_col,
                 roof_col=self.roof_col,
                 window_col=self.window_col,
                 facing_up=False)


# ──────────────────────────────────────────────────────────────
#  HUD — score, speed indicator
# ──────────────────────────────────────────────────────────────
class HUD:
    """Renders the on-screen score and speed display."""

    def __init__(self, font_big: pygame.font.Font,
                 font_small: pygame.font.Font) -> None:
        self.font_big   = font_big
        self.font_small = font_small
```

```
    def draw(self, surface: pygame.Surface,
             score: int, road_speed: float) -> None:
        # Semi-transparent score panel
        panel = pygame.Surface((160, 56), pygame.SRCALPHA)
        panel.fill((0, 0, 0, 140))
        surface.blit(panel, (10, 10))

        score_surf = self.font_big.render(f"{score:06d}", True, C_YELLOW)
        surface.blit(score_surf, (18, 14))

        km_h = int(road_speed * 18)          # fictitious km/h for flavour
        speed_surf = self.font_small.render(f"Speed: {km_h} km/h", True, C_WHITE)
        surface.blit(speed_surf, (18, 42))


# ──────────────────────────────────────────────────────────────
#  SCREEN HELPERS
# ──────────────────────────────────────────────────────────────
def draw_start_screen(surface: pygame.Surface,
                      font_title: pygame.font.Font,
                      font_body: pygame.font.Font) -> None:
    """Render the title / start screen."""
    # Dark overlay
    overlay = pygame.Surface((SCREEN_W, SCREEN_H), pygame.SRCALPHA)
    overlay.fill((0, 0, 0, 180))
    surface.blit(overlay, (0, 0))

    # Title
    title = font_title.render("CAR RACER", True, C_YELLOW)
    surface.blit(title, title.get_rect(center=(SCREEN_W // 2, 260)))

    # Subtitle
    sub = font_body.render("Dodge the traffic — survive!", True, C_WHITE)
    surface.blit(sub, sub.get_rect(center=(SCREEN_W // 2, 330)))

    # Controls info
    for i, line in enumerate([
        "← →  Arrow keys  –  Move left / right",
        "Press  ENTER  to start",
    ]):
        txt = font_body.render(line, True, (200, 200, 200))
        surface.blit(txt, txt.get_rect(center=(SCREEN_W // 2, 420 + i * 38)))
```

```
def draw_game_over_screen(surface: pygame.Surface,
                          font_title: pygame.font.Font,
                          font_body: pygame.font.Font,
                          score: int,
                          best_score: int) -> None:
    """Render the game-over screen with score summary."""
    overlay = pygame.Surface((SCREEN_W, SCREEN_H), pygame.SRCALPHA)
    overlay.fill((0, 0, 0, 190))
    surface.blit(overlay, (0, 0))

    # "GAME OVER"
    title = font_title.render("GAME OVER", True, C_RED)
    surface.blit(title, title.get_rect(center=(SCREEN_W // 2, 240)))

    # Scores
    score_txt = font_body.render(f"Score :  {score:06d}", True, C_WHITE)
    best_txt  = font_body.render(f"Best  :  {best_score:06d}", True, C_YELLOW)
    surface.blit(score_txt, score_txt.get_rect(center=(SCREEN_W // 2, 330)))
    surface.blit(best_txt,  best_txt.get_rect(center=(SCREEN_W // 2, 375)))

    # Restart prompt
    restart = font_body.render("Press  R  or  ENTER  to play again", True, (200, 200, 200))
    surface.blit(restart, restart.get_rect(center=(SCREEN_W // 2, 460)))

    quit_txt = font_body.render("ESC  —  Quit", True, (160, 160, 160))
    surface.blit(quit_txt, quit_txt.get_rect(center=(SCREEN_W // 2, 508)))


# ──────────────────────────────────────────────────────────────
#  GAME STATE  — holds everything for one run
# ──────────────────────────────────────────────────────────────
class GameState:
    """
    Encapsulates all mutable game data for a single play-through.
    Creating a new GameState resets everything cleanly.
    """

    def __init__(self) -> None:
        self.road      = Road()
        self.forest    = Forest()
        self.player    = Player()
        self.enemies:  list[Enemy] = []

        self.score       = 0
```

```
        self.road_speed   = float(INITIAL_ROAD_SPEED)
        self.spawn_timer  = 0       # counts frames until next enemy spawn
        self.spawn_interval = ENEMY_SPAWN_INTERVAL
        self.alive        = True

    def update(self, keys: pygame.key.ScancodeWrapper) -> None:
        """Advance everything by one frame."""
        if not self.alive:
            return

        # ── Gradually increase difficulty ──────────
        self.road_speed  += SPEED_INCREMENT
        self.spawn_interval = max(MIN_SPAWN_INTERVAL,
                                  ENEMY_SPAWN_INTERVAL - self.score // 800)

        # ── Scroll road and forest ─────────────────
        self.road.update(self.road_speed)
        self.forest.update(self.road_speed)

        # ── Move player ────────────────────────────
        self.player.handle_input(keys)

        # ── Spawn enemy cars ───────────────────────
        self.spawn_timer += 1
        if self.spawn_timer >= self.spawn_interval:
            self.spawn_timer = 0
            self.enemies.append(Enemy(self.road_speed))

        # ── Move enemies, remove off-screen ────
        for enemy in self.enemies:
            enemy.update()
        self.enemies = [e for e in self.enemies if not e.is_off_screen()]

        # ── Collision detection ────────────────────
        for enemy in self.enemies:
            if self.player.rect.colliderect(enemy.rect):
                self.alive = False
                return          # stop processing this frame

        # ── Score (only if alive) ──────────────────
        self.score += SCORE_PER_FRAME

    def draw(self, surface: pygame.Surface) -> None:
        """Render the full scene in correct back-to-front order."""
```

```
        # 1. Grass background
        surface.fill(C_SKY)

        # 2. Forest (behind road)
        self.forest.draw(surface)

        # 3. Road
        self.road.draw(surface)

        # 4. Enemy cars
        for enemy in self.enemies:
            enemy.draw(surface)

        # 5. Player car (on top of everything)
        self.player.draw(surface)


# ──────────────────────────────────────────────────────────────
#  MAIN  — game loop
# ──────────────────────────────────────────────────────────────
def main() -> None:
    pygame.init()
    pygame.display.set_caption("Car Racer")

    screen = pygame.display.set_mode((SCREEN_W, SCREEN_H))
    clock  = pygame.time.Clock()

    # Fonts
    font_title = pygame.font.SysFont("arial", 64, bold=True)
    font_body  = pygame.font.SysFont("arial", 26)
    font_score = pygame.font.SysFont("consolas", 26, bold=True)
    font_small = pygame.font.SysFont("consolas", 18)

    hud = HUD(font_score, font_small)

    # ── States: "start"  |  "playing"  |  "gameover" ──
    state      = "start"
    game       = GameState()      # pre-created so background shows on title
    best_score = 0

    # ── Main loop ──────────────────────────────────────────────
    while True:
        clock.tick(FPS)
        keys = pygame.key.get_pressed()
```

```
# ── Event handling ──────────────────────────────
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()

    if event.type == pygame.KEYDOWN:

        # Quit from anywhere
        if event.key == pygame.K_ESCAPE:
            pygame.quit()
            sys.exit()

        # Start game
        if state == "start":
            if event.key in (pygame.K_RETURN, pygame.K_KP_ENTER):
                game  = GameState()
                state = "playing"

        # Restart after game over
        elif state == "gameover":
            if event.key in (pygame.K_r, pygame.K_RETURN,
                             pygame.K_KP_ENTER):
                game  = GameState()
                state = "playing"

# ── Update logic ────────────────────────────────
if state == "playing":
    game.update(keys)
    if not game.alive:
        best_score = max(best_score, game.score)
        state = "gameover"

# ── Rendering ───────────────────────────────────
# Always draw the game world (visible behind overlays too)
game.draw(screen)

if state == "start":
    draw_start_screen(screen, font_title, font_body)

elif state == "playing":
    hud.draw(screen, game.score, game.road_speed)
```

```
        elif state == "gameover":
            hud.draw(screen, game.score, game.road_speed)
            draw_game_over_screen(screen, font_title, font_body,
                                  game.score, best_score)

        pygame.display.flip()


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