

```
// =====
// TETRIS AUTOPLAY for R1BBIT
// Waveshare ESP32-P4 Smart 86 Box, 720x720
//
// The game plays itself: for every new piece the AI tries all
// rotations and columns, scores each landing spot (height, holes,
// bumpiness, cleared lines), then rotates, moves and drops the piece.
// After GAME OVER a new game starts automatically in 3 seconds.
// =====

CONFIG
DISPLAY_DRIVER ST7703
DISPLAY_BUS DSI
DISPLAY_W 720
DISPLAY_H 720
DISPLAY_RST 27
DISPLAY_BL 26
END_CONFIG
DISPLAY_INIT
TIMER_RESET

// ----- settings -----
VAR BASEDROP = 270 // fall speed at level 1, ms per row (3x faster)
VAR MINDROP = 40 // fastest fall speed, ms per row
VAR MOVEMS = 35 // delay between AI moves (rotate / shift), ms
VAR REPAIRROWS = 1 // field rows re-checked per loop (background repair)
VAR CS = 30 // cell size in pixels
VAR FX = 30, FY = 60 // playing field origin (10x20 cells)
VAR BG = RGB565(10, 10, 25)
VAR EMPTYC = RGB565(28, 28, 55)
VAR WHITE = RGB565(255, 255, 255)
VAR GRAY = RGB565(150, 150, 180)
VAR REDC = RGB565(255, 60, 60)
VAR GREENC = RGB565(60, 255, 120)
// Colours: 0 = empty, 1..7 = I O T S Z J L
VAR COL[8]
COL[0] = EMPTYC
COL[1] = RGB565(0, 220, 255)
COL[2] = RGB565(255, 220, 0)
COL[3] = RGB565(180, 0, 255)
COL[4] = RGB565(0, 230, 70)
COL[5] = RGB565(255, 40, 40)
COL[6] = RGB565(40, 90, 255)
COL[7] = RGB565(255, 140, 0)

// ----- piece shapes -----
// Each shape is a 4x4 bit mask: bit number = y * 4 + x.
// Index = piece * 4 + rotation. Order: I, O, T, S, Z, J, L.
VAR MK[28]
MK[0] = 240
MK[1] = 17476
MK[2] = 3840
MK[3] = 8738
MK[4] = 102
MK[5] = 102
MK[6] = 102
MK[7] = 102
MK[8] = 114
```

```

MK[9] = 610
MK[10] = 624
MK[11] = 562
MK[12] = 54
MK[13] = 1122
MK[14] = 864
MK[15] = 561
MK[16] = 99
MK[17] = 612
MK[18] = 1584
MK[19] = 306
MK[20] = 113
MK[21] = 550
MK[22] = 1136
MK[23] = 802
MK[24] = 116
MK[25] = 1570
MK[26] = 368
MK[27] = 547
// Decoded cells: SH[(piece*4 + rot)8 + cell/2] = x, +1 = y
VAR SH[224]
VAR PIX, BI, M, NC
FOR PIX = 0 TO 27
M = MK[PIX]
NC = 0
FOR BI = 0 TO 15
IF M - FLOOR(M / 2) * 2 = 1 THEN
SH[PIX * 8 + NC * 2] = BI - FLOOR(BI / 4) * 4
SH[PIX * 8 + NC * 2 + 1] = FLOOR(BI / 4)
NC = NC + 1
ENDIF
M = FLOOR(M / 2)
NEXT
NEXT
VAR KICK[5]
KICK[0] = 0
KICK[1] = -1
KICK[2] = 1
KICK[3] = -2
KICK[4] = 2
// ----- state -----
VAR GRID[20, 10]
VAR HT[10], TH[10], LOW[10], ROWN[20], PRC[20], CXS[4], CYS[4]
VAR P, R, PX, PY, NP
VAR TP, TR, TXX, TYY, OK
VAR B, CX, CY
VAR DI, DCX, DCY, DCC, DCOL
VAR SCORE, LINES, LEVEL, DROPMS, LASTFALL, LASTMOVE, NOW, RESTART
VAR GAMEOVER, MOVED, ALIGNED, GAMES, BEST
VAR LR, LC, K, FULL, CLR, KI
VAR BESTR, BESTX, BESTS, AR, AX, NROT, VALID, SC
VAR EB, EI, ECX, ECY, LY, CAND, LN, HOLES, AGG, BUMP, HH
VAR OX[4], OY[4], NX[4], NY[4], DJ, KEEP
VAR RCX, RPTR, RN
GAMES = 0

```

```

RPTR = 0
BEST = 0
GOSUB DrawStatic
GOSUB NewGame
// ----- main loop -----
WHILE 1
NOW = TIMER_GET
IF GAMEOVER = 1 AND NOW >= RESTART THEN
GOSUB NewGame
ENDIF
IF GAMEOVER = 0 AND ALIGNED = 0 AND NOW - LASTMOVE >= MOVEMS THEN
LASTMOVE = NOW
GOSUB AutoStep
ENDIF
IF GAMEOVER = 0 AND ALIGNED = 1 AND NOW - LASTFALL >= DROPMS THEN
LASTFALL = NOW
GOSUB StepDown
ENDIF
IF GAMEOVER = 0 THEN
GOSUB RepairRows
ENDIF
PAUSE 5
WEND
// =====
// GAME LOGIC
// =====

NewGame:
FOR LR = 0 TO 19
FOR LC = 0 TO 9
GRID[LR, LC] = 0
NEXT
NEXT
GAMES = GAMES + 1
SCORE = 0
LINES = 0
LEVEL = 1
DROPMS = BASEDROP
GAMEOVER = 0
GOSUB CalcStats
GOSUB DrawField
GOSUB DrawStats
GOSUB RandNext
GOSUB SpawnPiece
LASTFALL = TIMER_GET
LASTMOVE = LASTFALL
RETURN

RandNext:
NP = FLOOR(RND(7)) - 1 // RND(7) -> 1..7, piece index 0..6
IF NP < 0 THEN
NP = 0
ENDIF
IF NP > 6 THEN
NP = 6
ENDIF

```

RETURN

SpawnPiece:

P = NP

GOSUB RandNext

R = 0

PX = 3

PY = 0

ALIGNED = 0

TP = P

TR = R

TXX = PX

TTY = PY

GOSUB Fits

IF OK = 0 THEN

GOSUB GameOver

ELSE

DCOL = P + 1

GOSUB DrawPiece

GOSUB PlanMove

ENDIF

GOSUB DrawNext

RETURN

// ---- collision test: TP, TR, TXX, TTY -> OK (1 = fits) ----

Fits:

OK = 1

B = (TP * 4 + TR) * 8

CX = TXX + SH[B]

CY = TTY + SH[B + 1]

GOSUB CheckCell

CX = TXX + SH[B + 2]

CY = TTY + SH[B + 3]

GOSUB CheckCell

CX = TXX + SH[B + 4]

CY = TTY + SH[B + 5]

GOSUB CheckCell

CX = TXX + SH[B + 6]

CY = TTY + SH[B + 7]

GOSUB CheckCell

RETURN

CheckCell:

IF CX < 0 OR CX > 9 OR CY > 19 THEN

OK = 0

ELSE IF CY >= 0 THEN

IF GRID[CY, CX] <> 0 THEN

OK = 0

ENDIF

ENDIF

RETURN

// ---- one AI move toward the planned rotation / column ----

AutoStep:

IF R <> BESTR THEN

```
GOSUB TryRotate
IF OK = 0 THEN
BESTR = R
ENDIF
ELSE IF PX < BESTX THEN
TXX = PX + 1
GOSUB TryShift
IF OK = 0 THEN
BESTX = PX
ENDIF
ELSE IF PX > BESTX THEN
TXX = PX - 1
GOSUB TryShift
IF OK = 0 THEN
BESTX = PX
ENDIF
ENDIF
IF R = BESTR AND PX = BESTX THEN
ALIGNED = 1
LASTFALL = NOW
ENDIF
RETURN
```

```
TryShift:
TP = P
TR = R
TTY = PY
GOSUB Fits
IF OK = 1 THEN
GOSUB SaveOld
PX = TXX
GOSUB MoveDraw
ENDIF
RETURN
```

```
TryRotate:
TP = P
TR = R + 1
IF TR > 3 THEN
TR = 0
ENDIF
TTY = PY
OK = 0
KI = 0
WHILE (KI < 5 AND OK = 0)
TXX = PX + KICK[KI]
GOSUB Fits
KI = KI + 1
WEND
IF OK = 1 THEN
GOSUB SaveOld
R = TR
PX = TXX
GOSUB MoveDraw
ENDIF
RETURN
```

```
StepDown:
MOVED = 0
TP = P
TR = R
TXX = PX
TTY = PY + 1
GOSUB Fits
IF OK = 1 THEN
GOSUB SaveOld
PY = PY + 1
GOSUB MoveDraw
MOVED = 1
ELSE
GOSUB LockPiece
ENDIF
RETURN
```

```
LockPiece:
B = (P * 4 + R) * 8
FOR DI = 0 TO 3
DCX = PX + SH[B + DI * 2]
DCY = PY + SH[B + DI * 2 + 1]
IF DCY >= 0 THEN
GRID[DCY, DCX] = P + 1
ENDIF
NEXT
SCORE = SCORE + 10
GOSUB ClearLines
GOSUB DrawStats
GOSUB SpawnPiece
RETURN
```

```
ClearLines:
CLR = 0
LR = 19
WHILE (LR >= 0)
FULL = 1
FOR LC = 0 TO 9
IF GRID[LR, LC] = 0 THEN
FULL = 0
ENDIF
NEXT
IF FULL = 1 AND LR > 0 THEN
CLR = CLR + 1
FOR K = LR TO 1 STEP -1
FOR LC = 0 TO 9
GRID[K, LC] = GRID[K - 1, LC]
NEXT
NEXT
FOR LC = 0 TO 9
GRID[0, LC] = 0
NEXT
ELSE
LR = LR - 1
ENDIF
WEND
```

```

IF CLR > 0 THEN
  LINES = LINES + CLR
IF CLR = 1 THEN
  SCORE = SCORE + 100 * LEVEL
ELSE IF CLR = 2 THEN
  SCORE = SCORE + 300 * LEVEL
ELSE IF CLR = 3 THEN
  SCORE = SCORE + 500 * LEVEL
ELSE
  SCORE = SCORE + 800 * LEVEL
ENDIF
LEVEL = FLOOR(LINES / 10) + 1
DROPMS = BASEDROP - (LEVEL - 1) * 25
IF DROPMS < MINDROP THEN
  DROPMS = MINDROP
ENDIF
GOSUB DrawField
ENDIF
GOSUB CalcStats
RETURN

```

```
// ---- column heights HT[] and filled cells per row ROWN[] ----
```

```

CalcStats:
FOR LC = 0 TO 9
  HT[LC] = 0
NEXT
FOR LR = 0 TO 19
  ROWN[LR] = 0
  FOR LC = 0 TO 9
    IF GRID[LR, LC] <> 0 THEN
      ROWN[LR] = ROWN[LR] + 1
    IF HT[LC] = 0 THEN
      HT[LC] = 20 - LR
    ENDIF
  ENDIF
NEXT
NEXT
RETURN

```

```

GameOver:
GAMEOVER = 1
IF SCORE > BEST THEN
  BEST = SCORE
ENDIF
GOSUB DrawStats
DISPLAY_FILL_RECT(FX + 20, FY + 240, 260, 120, RGB565(0, 0, 0))
DISPLAY_RECT(FX + 20, FY + 240, 260, 120, REDC)
DISPLAY_TEXT(FX + 78, FY + 265, "GAME OVER", REDC, 16)
DISPLAY_TEXT(FX + 51, FY + 312, "new game in 3 s", WHITE, 12)
RESTART = TIMER_GET + 3000
RETURN

```

```

// =====
// AI
// =====
// Tries every rotation and column, keeps the best -> BESTR, BESTX

```

```

PlanMove:
BESTS = 0 - 999999
BESTR = R
BESTX = PX
NROT = 4
IF P = 1 THEN
NROT = 1
ENDIF
FOR AR = 0 TO NROT - 1
FOR AX = -2 TO 9
GOSUB EvalPlace
IF VALID = 1 AND SC > BESTS THEN
BESTS = SC
BESTR = AR
BESTX = AX
ENDIF
NEXT
NEXT
RETURN

```

// Evaluates placing piece P with rotation AR at column AX -> VALID, SC

```

EvalPlace:
VALID = 1
EB = (P * 4 + AR) * 8
FOR EI = 0 TO 3
CXS[EI] = AX + SH[EB + EI * 2]
IF CXS[EI] < 0 OR CXS[EI] > 9 THEN
VALID = 0
ENDIF
NEXT
IF VALID = 1 THEN
TP = P
TR = AR
TXX = AX
TYY = 0
GOSUB Fits
VALID = OK
ENDIF
IF VALID = 1 THEN
GOSUB FindLanding
ENDIF
IF VALID = 1 THEN
GOSUB ScorePlace
ENDIF
RETURN

```

// Row where the piece lands when dropped straight down -> LY

```

FindLanding:
LY = 99
FOR EI = 0 TO 3
ECX = CXS[EI]
CAND = 19 - HT[ECX] - SH[EB + EI * 2 + 1]
IF CAND < LY THEN
LY = CAND
ENDIF

```

```

NEXT
IF LY < 0 THEN
VALID = 0
ENDIF
RETURN

// Heuristic: fewer height, holes and bumps; more cleared lines

ScorePlace:
FOR EI = 0 TO 9
TH[EI] = HT[EI]
LOW[EI] = -1
NEXT
FOR EI = 0 TO 3
ECX = CXS[EI]
ECY = LY + SH[EB + EI * 2 + 1]
CYS[EI] = ECY
PRC[ECY] = 0
HH = 20 - ECY
IF HH > TH[ECX] THEN
TH[ECX] = HH
ENDIF
IF ECY > LOW[ECX] THEN
LOW[ECX] = ECY
ENDIF
NEXT
FOR EI = 0 TO 3
ECY = CYS[EI]
PRC[ECY] = PRC[ECY] + 1
NEXT
LN = 0
FOR EI = 0 TO 3
ECY = CYS[EI]
IF PRC[ECY] > 0 AND ROWN[ECY] + PRC[ECY] = 10 THEN
LN = LN + 1
ENDIF
PRC[ECY] = 0
NEXT
HOLES = 0
AGG = 0
FOR EI = 0 TO 9
IF LOW[EI] >= 0 THEN
HOLES = HOLES + 19 - HT[EI] - LOW[EI]
ENDIF
AGG = AGG + TH[EI]
NEXT
BUMP = 0
FOR EI = 0 TO 8
BUMP = BUMP + ABS(TH[EI] - TH[EI + 1])
NEXT
AGG = AGG - LN * 10
SC = 0 - 51 * AGG + 76 * LN - 60 * HOLES - 18 * BUMP
RETURN

// =====
// DRAWING

```

```
// =====
```

DrawPiece:

B = (P * 4 + R) * 8

FOR DI = 0 TO 3

DCX = PX + SH[B + DI * 2]

DCY = PY + SH[B + DI * 2 + 1]

IF DCY >= 0 THEN

DISPLAY_FILL_RECT(FX + DCX * CS + 1, FY + DCY * CS + 1, CS - 2, CS - 2, COL[DCOL])

ENDIF

NEXT

RETURN

// ---- remember cells of the piece before it moves ----

SaveOld:

B = (P * 4 + R) * 8

FOR DI = 0 TO 3

OX[DI] = PX + SH[B + DI * 2]

OY[DI] = PY + SH[B + DI * 2 + 1]

NEXT

RETURN

// ---- redraw after a move ----

// Paint all new cells first, then clear only the old cells the piece

// no longer covers. Shared cells are never cleared, so the piece is

// never drawn "half-erased".

MoveDraw:

B = (P * 4 + R) * 8

FOR DI = 0 TO 3

NX[DI] = PX + SH[B + DI * 2]

NY[DI] = PY + SH[B + DI * 2 + 1]

NEXT

FOR DI = 0 TO 3

DCX = NX[DI]

DCY = NY[DI]

IF DCY >= 0 THEN

DISPLAY_FILL_RECT(FX + DCX * CS + 1, FY + DCY * CS + 1, CS - 2, CS - 2, COL[P + 1])

ENDIF

NEXT

FOR DI = 0 TO 3

KEEP = 0

FOR DJ = 0 TO 3

IF OX[DI] = NX[DJ] AND OY[DI] = NY[DJ] THEN

KEEP = 1

ENDIF

NEXT

DCX = OX[DI]

DCY = OY[DI]

IF KEEP = 0 AND DCY >= 0 THEN

DISPLAY_FILL_RECT(FX + DCX * CS + 1, FY + DCY * CS + 1, CS - 2, CS - 2, EMPTYC)

ENDIF

NEXT

RETURN

// ---- background repair ----

```
// Every loop, REPAIRROWS rows of the field are repainted from the real
// game state (placed blocks + current piece), going round the field.
// Any cell the display lost or left behind is fixed within a moment,
// whatever the cause.
```

RepairRows:

```
// put the current piece into GRID temporarily
B = (P * 4 + R) * 8
FOR DI = 0 TO 3
  DCX = PX + SH[B + DI * 2]
  DCY = PY + SH[B + DI * 2 + 1]
  IF DCY >= 0 THEN
    GRID[DCY, DCX] = P + 1
  ENDIF
NEXT
FOR RN = 1 TO REPAIRROWS
  FOR RCX = 0 TO 9
    DCC = GRID[RPTR, RCX]
    DISPLAY_FILL_RECT(FX + RCX * CS + 1, FY + RPTR * CS + 1, CS - 2, CS - 2, COL[DCC])
  NEXT
  RPTR = RPTR + 1
  IF RPTR > 19 THEN
    RPTR = 0
  ENDIF
NEXT
// take the piece back out of GRID
FOR DI = 0 TO 3
  DCX = PX + SH[B + DI * 2]
  DCY = PY + SH[B + DI * 2 + 1]
  IF DCY >= 0 THEN
    GRID[DCY, DCX] = 0
  ENDIF
NEXT
RETURN
```

DrawStatic:

```
DISPLAY_CLEAR(BG)
DISPLAY_TEXT(30, 18, "R1BBIT AI TETRIS", WHITE, 16)
DISPLAY_RECT(FX - 3, FY - 3, 10 * CS + 6, 20 * CS + 6, GRAY)
DISPLAY_TEXT(380, 70, "SCORE", GRAY, 12)
DISPLAY_TEXT(380, 150, "LINES", GRAY, 12)
DISPLAY_TEXT(560, 150, "LEVEL", GRAY, 12)
DISPLAY_TEXT(380, 230, "NEXT", GRAY, 12)
DISPLAY_RECT(377, 257, 126, 126, GRAY)
DISPLAY_TEXT(380, 420, "GAME", GRAY, 12)
DISPLAY_TEXT(560, 420, "BEST", GRAY, 12)
DISPLAY_TEXT(380, 560, "AI AUTOPLAY", GREENC, 16)
RETURN
```

DrawField:

```
FOR LR = 0 TO 19
  FOR LC = 0 TO 9
    DCC = GRID[LR, LC]
    DISPLAY_FILL_RECT(FX + LC * CS + 1, FY + LR * CS + 1, CS - 2, CS - 2, COL[DCC])
  NEXT
NEXT
```

RETURN

DrawStats:

```
DISPLAY_FILL_RECT(380, 92, 330, 32, BG)
DISPLAY_TEXT(380, 95, SCORE.0, WHITE, 16)
DISPLAY_FILL_RECT(380, 172, 330, 32, BG)
DISPLAY_TEXT(380, 175, LINES.0, WHITE, 16)
DISPLAY_TEXT(560, 175, LEVEL.0, WHITE, 16)
DISPLAY_FILL_RECT(380, 442, 330, 32, BG)
DISPLAY_TEXT(380, 445, GAMES.0, WHITE, 16)
DISPLAY_TEXT(560, 445, BEST.0, WHITE, 16)
RETURN
```

DrawNext:

```
DISPLAY_FILL_RECT(380, 260, 120, 120, BG)
B = (NP * 4) * 8
FOR DI = 0 TO 3
  DCX = SH[B + DI * 2]
  DCY = SH[B + DI * 2 + 1]
  DISPLAY_FILL_RECT(380 + DCX * CS + 1, 275 + DCY * CS + 1, CS - 2, CS - 2, COL[NP + 1])
NEXT
RETURN
```