Programming Fundamentals

Game Project: **ENDLESS RUNNER**

Submitted By:

1. Ans Rizwan — **24F-0779**

2. Muhammad Shaheer — **24F-0599**

# SOURCE CODE

1. //OUR GAME THE ENDLESS RUNNER FILLED WITH THRILL AND SUSPENSE

2. //DEVELOPERS: F24-0779 AND F24-0599

3. #include <iostream>

4. #include <string>

5. #include <conio.h>

6. #include <Windows.h>

7. #include <iomanip>

8. #include <fstream>

9. #include <ctime>

10. #include <cstdlib>

11. #include <mmsystem.h>

12. #include <sstream> // For string buffering

13.

14. using namespace std;

15.

16. #pragma comment(lib, "winmm.lib")

17.

18. // Game constants

19. const int MAP\_WIDTH = 40; // Width of game map

20. const int MAP\_HEIGHT = 20; // Height of game map

21. const float INITIAL\_SPEED = 1.0f; // Starting speed multiplier

22. const float MAX\_SPEED = 2.5f; // Max speed cap

23. const int COIN\_VALUE = 10; // Points per coin

24. const float SPEED\_INCREMENT = 0.1f; // Reduced speed increase per milestone

25. const int FIXED\_FRAME\_DELAY = 16; // Frame delay for ~60 FPS (ms)

26. const int MILESTONE\_1 = 200; // Distance milestones

27. const int MILESTONE\_2 = 400;

28. const int MILESTONE\_3 = 600;

29. const int MILESTONE\_4 = 800;

30. const int MILESTONE\_5 = 1000;

31. const int MILESTONE\_6 = 1200;

32. const int MILESTONE\_7 = 1400;

33. const int MILESTONE\_8 = 1600;

34. const int MILESTONE\_9 = 1800;

35. const int MILESTONE\_10 = 2000;

36. const int MILESTONE\_11 = 2500;

37. const int MILESTONE\_12 = 3000;

38. const int MILESTONE\_13 = 3500;

39. const int MILESTONE\_14 = 4000;

40. const int MILESTONE\_15 = 4500;

41. const int MILESTONE\_16 = 5000;

42. const int MILESTONE\_17 = 6000;

43. const int MILESTONE\_18 = 7000;

44. const int MILESTONE\_19 = 8000;

45. const int MILESTONE\_20 = 9000;

46. const int MILESTONE\_21 = 10000;

47. const int MAX\_OBSTACLES = 100; // Max obstacles on screen

48. const int MAX\_COINS = 100; // Max coins on screen

49. const int GRACE\_PERIOD = 3; // Frames of invulnerability after hit

50. const int MESSAGE\_START\_X = MAP\_WIDTH + 2; // Message display x-position

51. const int MESSAGE\_START\_Y = 2; // Message display y-position

52. const int MESSAGE\_WIDTH = 30; // Message area width

53. const int CONSOLE\_WIDTH = 80; // Console width for centering text

54. const int COMP\_INITIAL\_MOVE\_DELAY = 10; // Computer player move delay

55. const int COMP\_MAX\_SPEED = 3; // Max computer player speed

56.

57. // Player variables

58. int player\_x; // Player x-position

59. int player\_y; // Player y-position

60. int player\_lives; // Player lives

61. int player\_score; // Player score

62. int player\_distance; // Distance traveled

63. string player\_name; // Player name

64. int comp\_x; // Computer player x-position

65. int comp\_y; // Computer player y-position

66. bool comp\_active; // Computer player active flag

67. int comp\_move\_counter; // Counter for computer movement

68. int comp\_speed; // Computer speed

69.

70. // Message arrays

71. string message\_texts[5]; // Message texts

72. int message\_colors[5]; // Message colors

73. int message\_display\_times[5]; // Message display durations

74. int message\_start\_times[5]; // Message start times

75. bool message\_actives[5]; // Message active flags

76.

77. // Obstacle arrays

78. int obstacle\_xs[MAX\_OBSTACLES]; // Obstacle x-positions

79. int obstacle\_ys[MAX\_OBSTACLES]; // Obstacle y-positions

80. bool obstacle\_actives[MAX\_OBSTACLES]; // Obstacle active flags

81. int obstacle\_count; // Number of active obstacles

82.

83. // Coin arrays

84. int coin\_xs[MAX\_COINS]; // Coin x-positions

85. int coin\_ys[MAX\_COINS]; // Coin y-positions

86. bool coin\_collecteds[MAX\_COINS]; // Coin collected flags

87. bool coin\_actives[MAX\_COINS]; // Coin active flags

88. int coin\_count; // Number of active coins

89.

90. // Game state variables

91. int coin\_streak = 0; // Coin collection streak

92. int last\_coin\_time = 0; // Time of last coin collection

93. int last\_milestone = 0; // Last milestone reached

94. bool is\_jumping = false; // Jumping state

95. float jump\_velocity = 0.0f; // Jump velocity

96. float gravity = 0.5f; // Gravity for jump

97. float jump\_power = 1.5f; // Jump strength

98. float player\_y\_float; // Float y-position for smooth jumping

99. int grace\_frames = 0; // Invulnerability frames

100. float game\_speed = INITIAL\_SPEED; // Current speed multiplier

101.

102. // Set console text color

103. void set\_color(int color) {

104. SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);

105. }

106.

107. // Print centered text with color

108. void print\_centered(string text, int color = 7) {

109. int padding = (CONSOLE\_WIDTH - text.length()) / 2;

110. set\_color(color);

111. cout << setw(padding + text.length()) << text << endl;

112. }

113.

114. // Move cursor to (x, y)

115. void goto\_xy(int x, int y) {

116. COORD coord = { static\_cast<SHORT>(x), static\_cast<SHORT>(y) };

117. SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

118. }

119.

120. // Hide console cursor

121. void hide\_cursor() {

122. CONSOLE\_CURSOR\_INFO cursor = { 100, false };

123. SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor);

124. }

125.

126. // Set console window and buffer size

127. void set\_console\_size() {

128. HANDLE h\_console = GetStdHandle(STD\_OUTPUT\_HANDLE);

129. SMALL\_RECT window\_size = { 0, 0, MAP\_WIDTH + MESSAGE\_WIDTH + 5, MAP\_HEIGHT + 5 };

130. SetConsoleWindowInfo(h\_console, TRUE, &window\_size);

131. COORD buffer\_size = { MAP\_WIDTH + MESSAGE\_WIDTH + 5, MAP\_HEIGHT + 5 };

132. SetConsoleScreenBufferSize(h\_console, buffer\_size);

133. }

134.

135. // Show welcome screen

136. void show\_welcome\_screen() {

137. set\_console\_size();

138. system("cls");

139. set\_color(11);

140. cout << setfill('=') << setw(MAP\_WIDTH) << "=" << endl;

141. setfill(' ');

142.

143. // Display title

144. print\_centered(" \_\_\_\_\_\_ \_ \_ \_\_\_\_ \_ \_\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_\_ ", 10);

145. print\_centered("| \_\_\_\_| \\ | | \_ \\| | | \_\_\_\_|/ \_\_\_\_/ \_\_\_\_| ", 10);

146. print\_centered("| |\_\_ | \\| | | | | | | |\_\_ | (\_\_\_| (\_\_\_ ", 11);

147. print\_centered("| \_\_| | . ` | | | | | | \_\_| \\\_\_\_ \\\\\_\_\_ \\ ", 12);

148. print\_centered("| |\_\_\_\_| |\\ | |\_| | |\_\_\_\_| |\_\_\_\_ \_\_\_\_) |\_\_\_) | ", 14);

149. print\_centered("|\_\_\_\_\_\_|\_| \\\_|\_\_\_\_/|\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_/\_\_\_\_\_/ ", 12);

150.

151. print\_centered(" \_\_\_\_\_ \_ \_ \_ \_ \_ \_ \_\_\_\_\_\_ \_\_\_\_\_ ", 10);

152. print\_centered("| \_\_ \\| | | | \\ | | \\ | | \_\_\_\_| \_\_ \\ ", 10);

153. print\_centered("| |\_\_) | | | | \\| | \\| | |\_\_ | |\_\_) |", 11);

154. print\_centered("| \_ /| | | | . ` | . ` | \_\_| | \_ / ", 12);

155. print\_centered("| | \\ \\| |\_\_| | |\\ | |\\ | |\_\_\_\_| | \\ \\ ", 14);

156. print\_centered("|\_| \\\_\\\\\_\_\_\_/|\_| \\\_|\_| \\\_|\_\_\_\_\_\_|\_| \\\_\\", 12);

157.

158. cout << endl;

159. print\_centered("WELCOME TO THE ENDLESS RUNNER", 14);

160. print\_centered("Enter your Name and Press ENTER key to start...", 9);

161. set\_color(11);

162. cout << setfill('=') << setw(MAP\_WIDTH) << "=" << endl;

163. set\_color(7);

164. }

165.

166. // Add message to queue

167. void add\_message(string text, int color, int display\_time = 5000) {

168. int slot = -1;

169. // Find empty slot

170. for (int i = 0; i < 5; i++) {

171. if (!message\_actives[i]) {

172. slot = i;

173. break;

174. }

175. }

176.

177. // If no empty slot, replace oldest

178. if (slot == -1) {

179. int oldest\_time = GetTickCount();

180. int oldest\_index = 0;

181. for (int i = 0; i < 5; i++) {

182. if (message\_start\_times[i] < oldest\_time) {

183. oldest\_time = message\_start\_times[i];

184. oldest\_index = i;

185. }

186. }

187. slot = oldest\_index;

188. }

189.

190. // Set message properties

191. message\_texts[slot] = text;

192. message\_colors[slot] = color;

193. message\_display\_times[slot] = display\_time;

194. message\_start\_times[slot] = GetTickCount();

195. message\_actives[slot] = true;

196. }

197.

198. // Update message timers

199. void update\_messages() {

200. int current\_time = GetTickCount();

201. for (int i = 0; i < 5; i++) {

202. if (message\_actives[i] && current\_time - message\_start\_times[i] > message\_display\_times[i]) {

203. message\_actives[i] = false;

204. }

205. }

206. }

207.

208. // Display active messages

209. void display\_messages() {

210. // Clear message area

211. for (int y = 0; y < MAP\_HEIGHT; y++) {

212. goto\_xy(MESSAGE\_START\_X, y);

213. cout << string(MESSAGE\_WIDTH, ' ');

214. }

215.

216. // Draw active messages

217. int y\_offset = 0;

218. for (int i = 0; i < 5; i++) {

219. if (message\_actives[i]) {

220. goto\_xy(MESSAGE\_START\_X, MESSAGE\_START\_Y + y\_offset);

221. set\_color(message\_colors[i]);

222. cout << message\_texts[i];

223. set\_color(7);

224. y\_offset += 2;

225. }

226. }

227. }

228.

229. // Initialize message arrays

230. void initialize\_messages() {

231. for (int i = 0; i < 5; i++) {

232. message\_texts[i] = "";

233. message\_colors[i] = 7;

234. message\_display\_times[i] = 0;

235. message\_start\_times[i] = 0;

236. message\_actives[i] = false;

237. }

238. }

239.

240. // Message arrays for events

241. const string DISTANCE\_MESSAGES[] = {

242. "KEEP RUNNING!", "IMPRESSIVE DISTANCE!", "MARATHON RUNNER!",

243. "GOING THE DISTANCE!", "UNSTOPPABLE JOURNEY!", "ULTRAMARATHON STATUS!",

244. "TRANSCONTINENTAL RUNNER!", "BREAKING RECORDS!", "LEGENDARY DISTANCE!",

245. "COSMIC DISTANCE ACHIEVED!"

246. };

247. const string SPEED\_INCREASE\_MESSAGES[] = {

248. "SPEED BOOST!", "GETTING FASTER!", "TURBO MODE!", "BREAKING LIMITS!",

249. "HYPERSPEED ENGAGED!", "SUPERSONIC SPEED!", "WARP SPEED ACTIVATED!",

250. "TIME-SPACE DISTORTION!", "LIGHT SPEED ACHIEVED!", "BEYOND PHYSICS!"

251. };

252. const string LIVES\_MESSAGES[] = {

253. "LAST CHANCE!", "BE CAREFUL!", "STAY ALIVE!"

254. };

255. const string SCORE\_MESSAGES[] = {

256. "NICE GOING!", "AWESOME MOVES!", "YOU'RE ON FIRE!",

257. "INCREDIBLE SKILLS!", "UNSTOPPABLE!", "LEGENDARY PLAYER!"

258. };

259. const string COLLISION\_MESSAGES[] = {

260. "OUCH! THAT HURT!", "WATCH OUT!", "BOOM! COLLISION!",

261. "OH NO! YOU GOT HIT!", "CRASH! BE CAREFUL!"

262. };

263.

264. // Show collision message

265. void handle\_collision\_message() {

266. add\_message(COLLISION\_MESSAGES[rand() % 5], 12, 5000);

267. }

268.

269. // Show computer player activation message

270. void handle\_computer\_player\_message() {

271. add\_message("!!! DANGER !!!", 12, 8000);

272. add\_message("Computer Player Activated!", 14, 8000);

273. }

274.

275. // Check distance milestones

276. void check\_distance\_milestones() {

277. int milestones[] = {

278. MILESTONE\_1, MILESTONE\_3, MILESTONE\_5, MILESTONE\_7, MILESTONE\_9,

279. MILESTONE\_11, MILESTONE\_13, MILESTONE\_16, MILESTONE\_19, MILESTONE\_21

280. };

281. int message\_indices[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

282.

283. for (int i = 0; i < 10; i++) {

284. if (player\_distance == milestones[i]) {

285. add\_message(DISTANCE\_MESSAGES[message\_indices[i]], 11 + (i % 5), 5000);

286. break;

287. }

288. }

289. }

290.

291. // Show speed increase message

292. void display\_speed\_increase\_message(int milestone\_index) {

293. int message\_index = milestone\_index % 10;

294. int color\_index = 9 + (milestone\_index % 7);

295. add\_message(SPEED\_INCREASE\_MESSAGES[message\_index], color\_index, 4000);

296. }

297.

298. // Warn when lives are low

299. void check\_lives\_warning() {

300. if (player\_lives == 1) {

301. static int flash\_counter = 0;

302. flash\_counter = (flash\_counter + 1) % 10;

303. if (flash\_counter == 0) {

304. add\_message(LIVES\_MESSAGES[rand() % 3], 12, 3000);

305. }

306. }

307. }

308.

309. // Show coin streak message

310. void display\_coin\_streak\_message(int coins\_collected) {

311. if (coins\_collected >= 3) {

312. add\_message("COIN STREAK: " + to\_string(coins\_collected) + "x!", 14, 3000);

313. }

314. }

315.

316. // Check score milestones

317. void check\_score\_milestones(int previous\_score, int current\_score) {

318. if (previous\_score < 50 && current\_score >= 50) {

319. add\_message(SCORE\_MESSAGES[0], 10, 5000);

320. }

321. else if (previous\_score < 100 && current\_score >= 100) {

322. add\_message(SCORE\_MESSAGES[1], 11, 5000);

323. }

324. else if (previous\_score < 200 && current\_score >= 200) {

325. add\_message(SCORE\_MESSAGES[2], 14, 5000);

326. }

327. else if (previous\_score < 300 && current\_score >= 300) {

328. add\_message(SCORE\_MESSAGES[3], 13, 5000);

329. }

330. else if (previous\_score < 500 && current\_score >= 500) {

331. add\_message(SCORE\_MESSAGES[4], 12, 5000);

332. }

333. else if (previous\_score < 1000 && current\_score >= 1000) {

334. add\_message(SCORE\_MESSAGES[5], 15, 5000);

335. }

336. }

337.

338. // Initialize game state

339. void initialize\_game() {

340. // Set player starting position

341. player\_x = MAP\_WIDTH / 2;

342. player\_y = MAP\_HEIGHT - 2;

343. player\_y\_float = static\_cast<float>(player\_y);

344. player\_lives = 3;

345. player\_score = 0;

346. player\_distance = 0;

347. is\_jumping = false;

348. jump\_velocity = 0.0f;

349. grace\_frames = 0;

350.

351. // Set computer player

352. comp\_x = MAP\_WIDTH / 2;

353. comp\_y = MAP\_HEIGHT - 10;

354. comp\_active = false;

355. comp\_move\_counter = 0;

356. comp\_speed = COMP\_INITIAL\_MOVE\_DELAY;

357.

358. // Reset game state

359. coin\_streak = 0;

360. last\_coin\_time = 0;

361. last\_milestone = 0;

362. game\_speed = INITIAL\_SPEED;

363.

364. // Initialize obstacles

365. obstacle\_count = 0;

366. for (int i = 0; i < 12 && obstacle\_count < MAX\_OBSTACLES; i++) {

367. obstacle\_xs[obstacle\_count] = 1 + rand() % (MAP\_WIDTH - 2);

368. obstacle\_ys[obstacle\_count] = rand() % (MAP\_HEIGHT - 10);

369. obstacle\_actives[obstacle\_count] = true;

370. obstacle\_count++;

371. }

372.

373. // Initialize coins

374. coin\_count = 0;

375. for (int i = 0; i < 20 && coin\_count < MAX\_COINS; i++) {

376. coin\_xs[coin\_count] = 1 + rand() % (MAP\_WIDTH - 2);

377. coin\_ys[coin\_count] = rand() % MAP\_HEIGHT;

378. coin\_collecteds[coin\_count] = false;

379. coin\_actives[coin\_count] = true;

380. coin\_count++;

381. }

382. }

383.

384. // Update game map

385. void update\_map(char map[MAP\_HEIGHT][MAP\_WIDTH]) {

386. // Draw borders and clear map

387. for (int y = 0; y < MAP\_HEIGHT; y++) {

388. for (int x = 0; x < MAP\_WIDTH; x++) {

389. if (x == 0 || x == MAP\_WIDTH - 1) map[y][x] = '|';

390. else if (y == 0 || y == MAP\_HEIGHT - 1) map[y][x] = '-';

391. else map[y][x] = ' ';

392. }

393. }

394. }

395.

396. // Draw game map with buffering

397. void draw\_map(char map[MAP\_HEIGHT][MAP\_WIDTH]) {

398. char temp\_map[MAP\_HEIGHT][MAP\_WIDTH];

399. memcpy(temp\_map, map, sizeof(temp\_map));

400.

401. // Place player and computer

402. temp\_map[player\_y][player\_x] = 'P';

403. if (comp\_active) temp\_map[comp\_y][comp\_x] = 'C';

404.

405. // Place obstacles

406. for (int i = 0; i < obstacle\_count; i++) {

407. if (obstacle\_actives[i] && obstacle\_ys[i] >= 0 && obstacle\_ys[i] < MAP\_HEIGHT &&

408. obstacle\_xs[i] >= 0 && obstacle\_xs[i] < MAP\_WIDTH) {

409. temp\_map[obstacle\_ys[i]][obstacle\_xs[i]] = 'X';

410. }

411. }

412.

413. // Place coins

414. for (int i = 0; i < coin\_count; i++) {

415. if (coin\_actives[i] && !coin\_collecteds[i] && coin\_ys[i] >= 0 && coin\_ys[i] < MAP\_HEIGHT &&

416. coin\_xs[i] >= 0 && coin\_xs[i] < MAP\_WIDTH) {

417. temp\_map[coin\_ys[i]][coin\_xs[i]] = 'O';

418. }

419. }

420.

421. // Buffer output to reduce cout calls

422. stringstream buffer;

423. for (int y = 0; y < MAP\_HEIGHT; y++) {

424. for (int x = 0; x < MAP\_WIDTH; x++) {

425. buffer << temp\_map[y][x];

426. }

427. buffer << '\n';

428. }

429.

430. // Draw buffered map

431. goto\_xy(0, 0);

432. set\_color(7); // Reset color

433. cout << buffer.str();

434.

435. // Apply colors (minimized calls)

436. for (int y = 0; y < MAP\_HEIGHT; y++) {

437. for (int x = 0; x < MAP\_WIDTH; x++) {

438. char ch = temp\_map[y][x];

439. if (ch == 'P' || ch == 'X' || ch == 'O' || ch == 'C' || ch == '|' || ch == '-') {

440. goto\_xy(x, y);

441. switch (ch) {

442. case 'P': set\_color(10); break;

443. case 'X': set\_color(12); break;

444. case 'O': set\_color(14); break;

445. case 'C': set\_color(9); break;

446. case '|': set\_color(12); break;

447. case '-': set\_color(11); break;

448. }

449. cout << ch;

450. }

451. }

452. }

453. set\_color(7);

454. }

455.

456. // Move player based on input

457. void move\_player(char key, char map[MAP\_HEIGHT][MAP\_WIDTH]) {

458. if (key == 'a' || key == 'A') {

459. if (player\_x - 1 > 1) player\_x -= 1;

460. else player\_x = 1; // Keep within left bound

461. }

462. else if (key == 'd' || key == 'D') {

463. if (player\_x + 1 < MAP\_WIDTH - 2) player\_x += 1;

464. else player\_x = MAP\_WIDTH - 2; // Keep within right bound

465. }

466.

467. if (!is\_jumping) {

468. if (key == 'w' || key == 'W') {

469. if (player\_y - 1 > 1) player\_y -= 1;

470. else player\_y = 1; // Keep within top bound

471. player\_y\_float = static\_cast<float>(player\_y);

472. }

473. else if (key == 's' || key == 'S') {

474. if (player\_y + 1 < MAP\_HEIGHT - 2) player\_y += 1;

475. else player\_y = MAP\_HEIGHT - 2; // Keep within bottom bound

476. player\_y\_float = static\_cast<float>(player\_y);

477. }

478. else if (key == ' ') {

479. is\_jumping = true;

480. jump\_velocity = -jump\_power;

481. }

482. }

483. }

484.

485. // Process player jump

486. void process\_jump() {

487. if (is\_jumping) {

488. jump\_velocity += gravity;

489. player\_y\_float += jump\_velocity;

490. int new\_y = static\_cast<int>(player\_y\_float);

491.

492. if (jump\_velocity < 0) {

493. player\_y = new\_y;

494. if (new\_y <= 1) {

495. player\_y = 1;

496. player\_y\_float = 1.0f;

497. jump\_velocity = 0;

498. is\_jumping = false;

499. }

500. }

501. else {

502. player\_y = new\_y;

503. player\_y\_float = static\_cast<float>(new\_y);

504. is\_jumping = false;

505. jump\_velocity = 0;

506. }

507. }

508. }

509.

510. // Move computer player

511. void move\_computer\_player(float game\_speed) {

512. int speed\_boost = 1 + static\_cast<int>(game\_speed - 1.0f);

513. if (speed\_boost > COMP\_MAX\_SPEED) speed\_boost = COMP\_MAX\_SPEED; // Cap computer speed

514.

515. // Chase player with some randomness

516. if (rand() % 100 < 65) {

517. if (comp\_x < player\_x) comp\_x += speed\_boost;

518. else if (comp\_x > player\_x) comp\_x -= speed\_boost;

519. if (comp\_y < player\_y) comp\_y += speed\_boost;

520. else if (comp\_y > player\_y) comp\_y -= speed\_boost;

521. }

522. else {

523. comp\_x += (rand() % 3) - 1;

524. comp\_y += (rand() % 3) - 1;

525. }

526.

527. // Keep within bounds

528. if (comp\_x < 1) comp\_x = 1;

529. else if (comp\_x > MAP\_WIDTH - 2) comp\_x = MAP\_WIDTH - 2;

530. if (comp\_y < 1) comp\_y = 1;

531. else if (comp\_y > MAP\_HEIGHT - 2) comp\_y = MAP\_HEIGHT - 2;

532. }

533.

534. // Check for collisions

535. bool check\_collision() {

536. if (!is\_jumping && grace\_frames == 0) {

537. for (int i = 0; i < obstacle\_count; i++) {

538. if (obstacle\_actives[i] && player\_x == obstacle\_xs[i] && player\_y == obstacle\_ys[i]) {

539. return true;

540. }

541. }

542. }

543. return comp\_active && player\_x == comp\_x && player\_y == comp\_y;

544. }

545.

546. // Collect coins

547. void collect\_coins() {

548. int current\_time = GetTickCount();

549. for (int i = 0; i < coin\_count; i++) {

550. if (coin\_actives[i] && !coin\_collecteds[i] && player\_x == coin\_xs[i] && player\_y == coin\_ys[i]) {

551. coin\_collecteds[i] = true;

552. player\_score += COIN\_VALUE;

553.

554. // Update streak

555. if (current\_time - last\_coin\_time < 2000) {

556. coin\_streak++;

557. }

558. else {

559. coin\_streak = 1;

560. }

561. last\_coin\_time = current\_time;

562. if (coin\_streak >= 3) display\_coin\_streak\_message(coin\_streak);

563. }

564. }

565. }

566.

567. // Generate new obstacles

568. void generate\_obstacles() {

569. int num\_new = 1 + rand() % 3;

570. for (int i = 0; i < num\_new && obstacle\_count < MAX\_OBSTACLES; i++) {

571. obstacle\_xs[obstacle\_count] = 1 + rand() % (MAP\_WIDTH - 2);

572. obstacle\_ys[obstacle\_count] = 1;

573. obstacle\_actives[obstacle\_count] = true;

574. obstacle\_count++;

575. }

576. }

577.

578. // Generate new coins

579. void generate\_coins() {

580. if (coin\_count < 95) {

581. int num\_new = 2 + rand() % 4;

582. for (int i = 0; i < num\_new && coin\_count < MAX\_COINS; i++) {

583. coin\_xs[coin\_count] = 1 + rand() % (MAP\_WIDTH - 2);

584. coin\_ys[coin\_count] = 1;

585. coin\_collecteds[coin\_count] = false;

586. coin\_actives[coin\_count] = true;

587. coin\_count++;

588. // Debug: Uncomment to confirm coin generation

589. // add\_message("Generated " + to\_string(num\_new) + " coins", 14, 1000);

590. }

591. }

592. }

593.

594. // Move obstacles and coins

595. void move\_game\_elements() {

596. static float accumulated\_speed = 0.0f;

597. accumulated\_speed += game\_speed;

598.

599. // Move elements when threshold reached

600. if (accumulated\_speed >= 4.0f) { // Increased to slow movement

601. for (int i = 0; i < obstacle\_count; i++) {

602. if (obstacle\_actives[i]) {

603. obstacle\_ys[i]++;

604. if (obstacle\_ys[i] >= MAP\_HEIGHT) obstacle\_actives[i] = false;

605. }

606. }

607. for (int i = 0; i < coin\_count; i++) {

608. if (coin\_actives[i] && !coin\_collecteds[i]) {

609. coin\_ys[i]++;

610. if (coin\_ys[i] >= MAP\_HEIGHT) coin\_actives[i] = false;

611. }

612. }

613. accumulated\_speed -= 4.0f;

614. }

615. }

616.

617. // Generate obstacles and coins

618. void generate\_game\_elements() {

619. static float obstacle\_accumulator = 0.0f;

620. static float coin\_accumulator = 0.0f;

621. obstacle\_accumulator += game\_speed;

622. coin\_accumulator += game\_speed;

623.

624. // Generate obstacles

625. if (obstacle\_accumulator >= 12.0f) { // Increased to reduce spawn frequency

626. generate\_obstacles();

627. obstacle\_accumulator -= 12.0f;

628. }

629.

630. // Generate coins

631. if (coin\_accumulator >= 10.0f) { // Increased to reduce spawn frequency

632. generate\_coins();

633. coin\_accumulator -= 10.0f;

634. }

635. }

636.

637. // Clean up inactive obstacles and coins

638. void cleanup\_game\_elements() {

639. // Clean obstacles

640. int new\_obstacle\_count = 0;

641. for (int i = 0; i < obstacle\_count; i++) {

642. if (obstacle\_actives[i]) {

643. obstacle\_xs[new\_obstacle\_count] = obstacle\_xs[i];

644. obstacle\_ys[new\_obstacle\_count] = obstacle\_ys[i];

645. obstacle\_actives[new\_obstacle\_count] = true;

646. new\_obstacle\_count++;

647. }

648. }

649. obstacle\_count = new\_obstacle\_count;

650.

651. // Clean coins

652. int new\_coin\_count = 0;

653. for (int i = 0; i < coin\_count; i++) {

654. if (coin\_actives[i] && !coin\_collecteds[i]) {

655. coin\_xs[new\_coin\_count] = coin\_xs[i];

656. coin\_ys[new\_coin\_count] = coin\_ys[i];

657. coin\_collecteds[new\_coin\_count] = false;

658. coin\_actives[new\_coin\_count] = true;

659. new\_coin\_count++;

660. }

661. }

662. coin\_count = new\_coin\_count;

663. }

664.

665. // Adjust game speed at milestones

666. void adjust\_game\_speed() {

667. int milestones[] = {

668. MILESTONE\_1, MILESTONE\_2, MILESTONE\_3, MILESTONE\_4, MILESTONE\_5,

669. MILESTONE\_6, MILESTONE\_7, MILESTONE\_8, MILESTONE\_9, MILESTONE\_10,

670. MILESTONE\_11, MILESTONE\_12, MILESTONE\_13, MILESTONE\_14, MILESTONE\_15,

671. MILESTONE\_16, MILESTONE\_17, MILESTONE\_18, MILESTONE\_19, MILESTONE\_20,

672. MILESTONE\_21

673. };

674. for (int i = 0; i < 21; i++) {

675. if (player\_distance >= milestones[i] && last\_milestone < milestones[i]) {

676. game\_speed += SPEED\_INCREMENT;

677. if (game\_speed > MAX\_SPEED) game\_speed = MAX\_SPEED; // Cap speed

678. last\_milestone = milestones[i];

679. display\_speed\_increase\_message(i);

680. break;

681. }

682. }

683. }

684.

685. // Save high score

686. void save\_high\_score() {

687. const int MAX\_HIGH\_SCORES = 5;

688. string names[MAX\_HIGH\_SCORES];

689. int scores[MAX\_HIGH\_SCORES];

690. int count = 0;

691.

692. // Read existing scores

693. ifstream score\_file("highscores.txt");

694. string name;

695. int value;

696. while (score\_file >> name >> value && count < MAX\_HIGH\_SCORES) {

697. names[count] = name;

698. scores[count] = value;

699. count++;

700. }

701. score\_file.close();

702.

703. // Add new score

704. if (count < MAX\_HIGH\_SCORES) {

705. names[count] = player\_name;

706. scores[count] = player\_score;

707. count++;

708. }

709. else {

710. for (int i = 0; i < MAX\_HIGH\_SCORES; i++) {

711. if (player\_score > scores[i]) {

712. for (int j = MAX\_HIGH\_SCORES - 1; j > i; j--) {

713. names[j] = names[j - 1];

714. scores[j] = scores[j - 1];

715. }

716. names[i] = player\_name;

717. scores[i] = player\_score;

718. break;

719. }

720. }

721. }

722.

723. // Sort scores

724. for (int i = 0; i < count; i++) {

725. for (int j = i + 1; j < count; j++) {

726. if (scores[j] > scores[i]) {

727. swap(scores[i], scores[j]);

728. swap(names[i], names[j]);

729. }

730. }

731. }

732.

733. // Save scores

734. ofstream out\_file("highscores.txt");

735. for (int i = 0; i < count; i++) {

736. out\_file << names[i] << " " << scores[i] << endl;

737. }

738. out\_file.close();

739. }

740.

741. // Display high scores

742. void display\_high\_scores() {

743. cout << "\nHigh Scores:\n------------\n";

744. ifstream score\_file("highscores.txt");

745. if (!score\_file) {

746. cout << "No high scores yet!\n";

747. return;

748. }

749.

750. string name;

751. int score;

752. int rank = 1;

753. while (score\_file >> name >> score && rank <= 5) {

754. cout << rank << ". " << name << ": " << score << endl;

755. rank++;

756. }

757. score\_file.close();

758. }

759.

760. int main() {

761. // Initialize audio

762. char mci\_buffer[256];

763. string music\_relative\_path = ".\\MUSIC\\music1.mp3";

764. bool audio\_playing = false;

765.

766. srand(static\_cast<unsigned int>(time(0)));

767. hide\_cursor();

768.

769. char map[MAP\_HEIGHT][MAP\_WIDTH];

770. bool game\_over = false;

771. char key;

772. int frame\_counter = 0;

773.

774. // Show welcome and get name

775. show\_welcome\_screen();

776. cout << "Enter your name: ";

777. getline(cin, player\_name);

778. Beep(800, 800);

779.

780. // Initialize game

781. initialize\_game();

782. initialize\_messages();

783. system("cls");

784.

785. // Try playing audio

786. string mp3\_command = "open \"" + music\_relative\_path + "\" type mpegvideo alias gamemusic";

787. if (mciSendString(mp3\_command.c\_str(), mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

788. if (mciSendString("play gamemusic repeat", mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

789. audio\_playing = true;

790. cout << "Playing MP3 music..." << endl;

791. }

792. }

793.

794. if (!audio\_playing) {

795. string wav\_relative\_path = ".\\MUSIC\\music.wav";

796. string wav\_command = "open \"" + wav\_relative\_path + "\" type mpegvideo alias gamemusic";

797. if (mciSendString(wav\_command.c\_str(), mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

798. if (mciSendString("play gamemusic repeat", mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

799. audio\_playing = true;

800. cout << "Playing WAV music..." << endl;

801. }

802. }

803. }

804.

805. if (!audio\_playing) {

806. char exe\_path[MAX\_PATH];

807. GetModuleFileName(NULL, exe\_path, MAX\_PATH);

808. string exe\_path\_str(exe\_path);

809. size\_t last\_slash\_pos = exe\_path\_str.find\_last\_of('\\');

810. string exe\_dir = last\_slash\_pos != string::npos ? exe\_path\_str.substr(0, last\_slash\_pos + 1) : "";

811. string full\_path = exe\_dir + "MUSIC\\music1.mp3";

812. string cmd = "open \"" + full\_path + "\" type mpegvideo alias gamemusic";

813.

814. if (mciSendString(cmd.c\_str(), mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

815. if (mciSendString("play gamemusic repeat", mci\_buffer, sizeof(mci\_buffer), NULL) == 0) {

816. audio\_playing = true;

817. cout << "Playing music from executable path..." << endl;

818. }

819. }

820. }

821.

822. if (!audio\_playing) {

823. DWORD error = GetLastError();

824. mciGetErrorString(error, mci\_buffer, sizeof(mci\_buffer));

825. cerr << "Failed to play audio. Error: " << error << " - " << mci\_buffer << endl;

826. }

827.

828. // Main game loop

829. int previous\_score = player\_score;

830. while (!game\_over) {

831. frame\_counter++;

832.

833. // Update game state

834. update\_map(map);

835. if (is\_jumping) process\_jump();

836. if (grace\_frames > 0) grace\_frames--;

837. draw\_map(map);

838. update\_messages();

839. display\_messages();

840. check\_distance\_milestones();

841. check\_lives\_warning();

842.

843. // Display stats

844. goto\_xy(0, MAP\_HEIGHT + 1);

845. cout << "Player: " << player\_name << " Lives: " << player\_lives

846. << " Score: " << player\_score << " Distance: " << player\_distance

847. << " Speed: " << fixed << setprecision(1) << game\_speed << "x" << endl;

848.

849. // Handle input

850. if (\_kbhit()) {

851. key = \_getch();

852. move\_player(key, map);

853. if (key == 'q' || key == 'Q') game\_over = true;

854. }

855.

856. // Move computer player

857. if (comp\_active) {

858. comp\_move\_counter++;

859. int move\_delay = COMP\_INITIAL\_MOVE\_DELAY - static\_cast<int>(game\_speed - 1.0f);

860. if (move\_delay < 3) move\_delay = 3; // Ensure minimum delay

861. if (comp\_move\_counter >= move\_delay) {

862. move\_computer\_player(game\_speed);

863. comp\_move\_counter = 0;

864. }

865. }

866.

867. // Check collisions

868. if (check\_collision()) {

869. player\_lives--;

870. handle\_collision\_message();

871. player\_x = MAP\_WIDTH / 2;

872. player\_y = MAP\_HEIGHT - 2;

873. player\_y\_float = static\_cast<float>(player\_y);

874. is\_jumping = false;

875. jump\_velocity = 0.0f;

876. grace\_frames = GRACE\_PERIOD;

877. if (player\_lives <= 0) game\_over = true;

878. }

879.

880. // Update game

881. collect\_coins();

882. check\_score\_milestones(previous\_score, player\_score);

883. previous\_score = player\_score;

884. player\_distance += 5;

885.

886. // Activate computer player

887. if (player\_distance >= MILESTONE\_10 && !comp\_active) {

888. comp\_active = true;

889. handle\_computer\_player\_message();

890. }

891.

892. // Update speed and elements

893. adjust\_game\_speed();

894. move\_game\_elements();

895. generate\_game\_elements();

896.

897. // Clean up every 10 frames to reduce lag

898. if (frame\_counter % 10 == 0) cleanup\_game\_elements();

899.

900. Sleep(FIXED\_FRAME\_DELAY);

901. }

902.

903. // Show game over screen

904. system("cls");

905. cout << "Game Over!\nPlayer: " << player\_name

906. << "\nScore: " << player\_score

907. << "\nDistance: " << player\_distance << "m" << endl;

908.

909. // Save and show high scores

910. save\_high\_score();

911. display\_high\_scores();

912.

913. // Clean up audio

914. if (audio\_playing) mciSendString(TEXT("close gamemusic"), NULL, 0, NULL);

915. system("pause");

916. return 0;

917. }

918.

Screenshots of the Game:

## 1. Game Start Display

A screenshot of a computer

AI-generated content may be incorrect.

## 2. Game Map with Player Info, Distance & Coins

A screen shot of a computer

AI-generated content may be incorrect.

## 3. Ahead Stages

A screen shot of a computer

AI-generated content may be incorrect.

## 4. Arrival of the Captain (ENEMY)

A screenshot of a computer game

AI-generated content may be incorrect.

## 6. High Scores

A black screen with white text

AI-generated content may be incorrect.