**本项目由我与计科2205班裘凯悦合作完成。**

在我们的汉诺塔游戏源码中，Execute() 方法是实现汉诺塔游戏关键算法的核心部分，通过递归来移动盘子。

1. void Execute(int n, int a, int b, int c)

2. {

3. if (n == 1)

4. {

5. // 如果只有一个盘子，直接将位于柱子 a 上的盘子移动到柱子 c

6. move(a, c);

7. }

8. else

9. {

10. // 如果有多个盘子

11. // 1. 将前 n-1 个盘子从柱子 a 借助柱子 c 移动到柱子 b

12. Execute(n - 1, a, c, b);

13.

14. // 2. 将位于柱子 a 上的最后一个盘子移动到柱子 c

15. move(a, c);

16.

17. // 3. 将柱子 b 上的 n-1 个盘子借助柱子 a 移动到柱子 c

18. Execute(n - 1, b, a, c);

19. }

20. }

21.

函数形参：

n：表示当前需要移动的盘子数量。

a、b、c：分别表示三个柱子的编号，其中盘子初始时位于柱子 a，目标是将其移动到柱子 c。

函数实现：

如果 n == 1，表示只有一个盘子，直接将柱子 a 上的盘子移动到柱子 c。

如果 n > 1，表示有多个盘子，按照汉诺塔的规则进行移动：

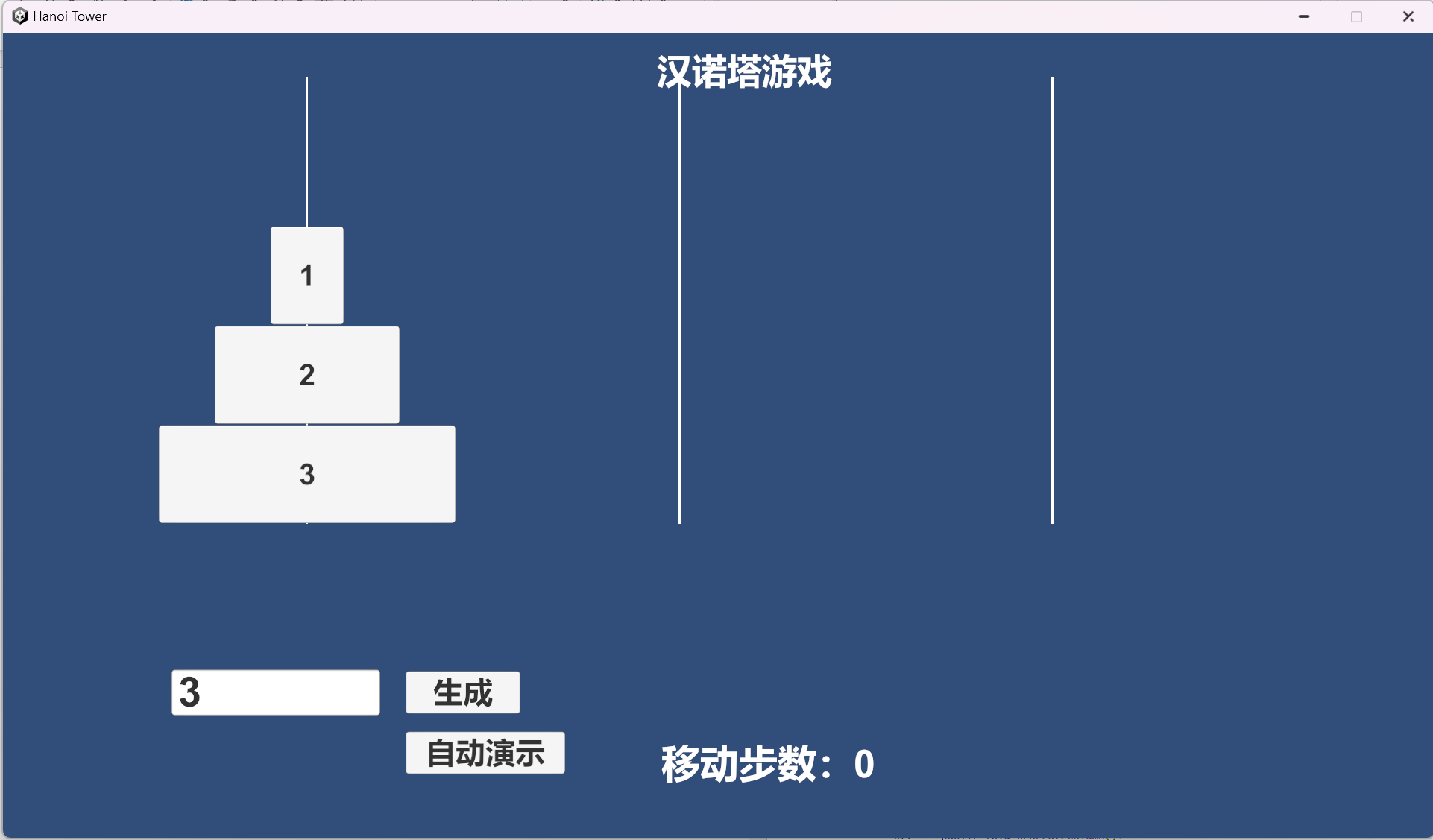
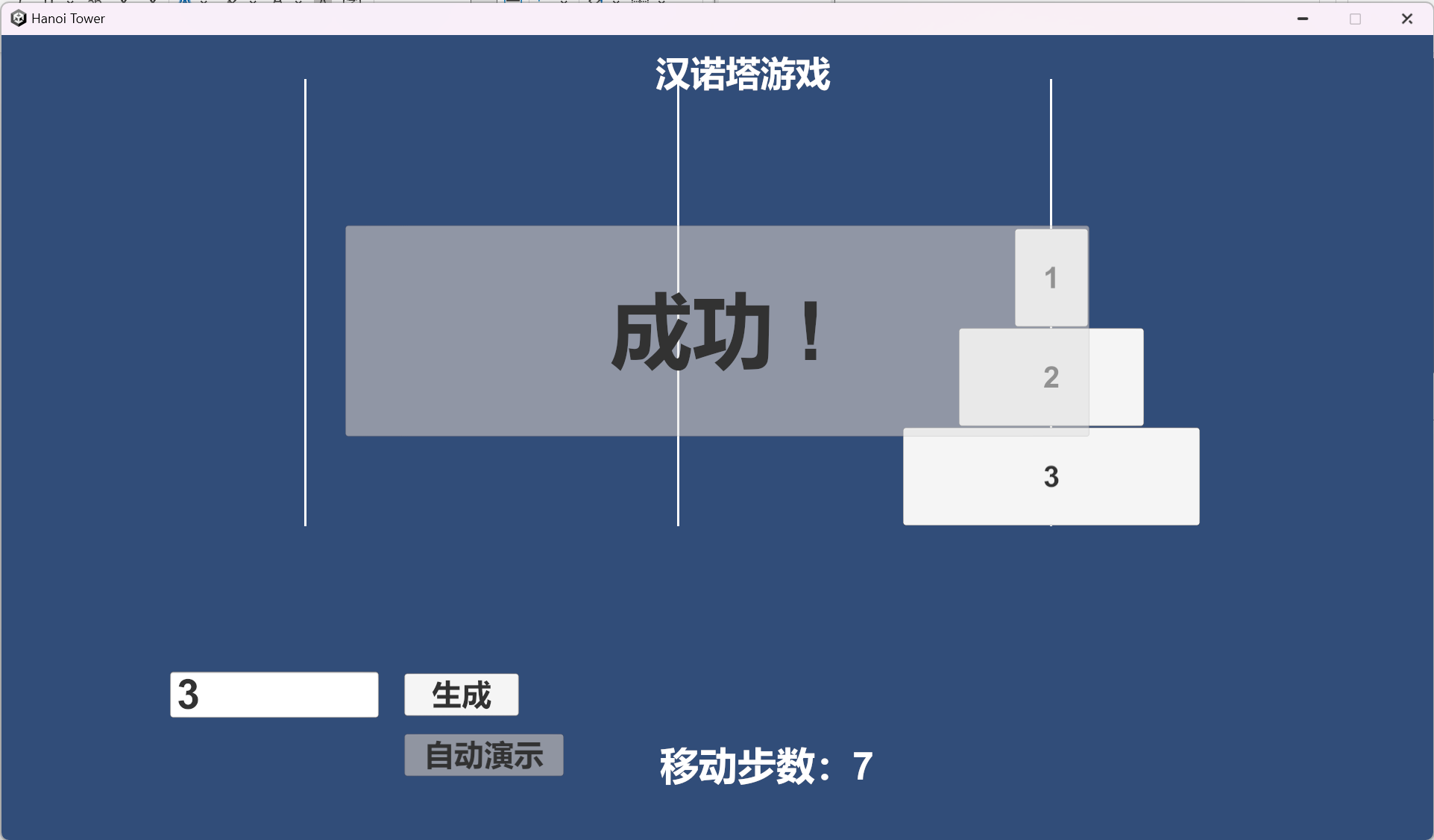
将前 n-1 个盘子从柱子 a 借助柱子 c 移动到柱子 b。

将位于柱子 a 上的最后一个盘子移动到柱子 c。

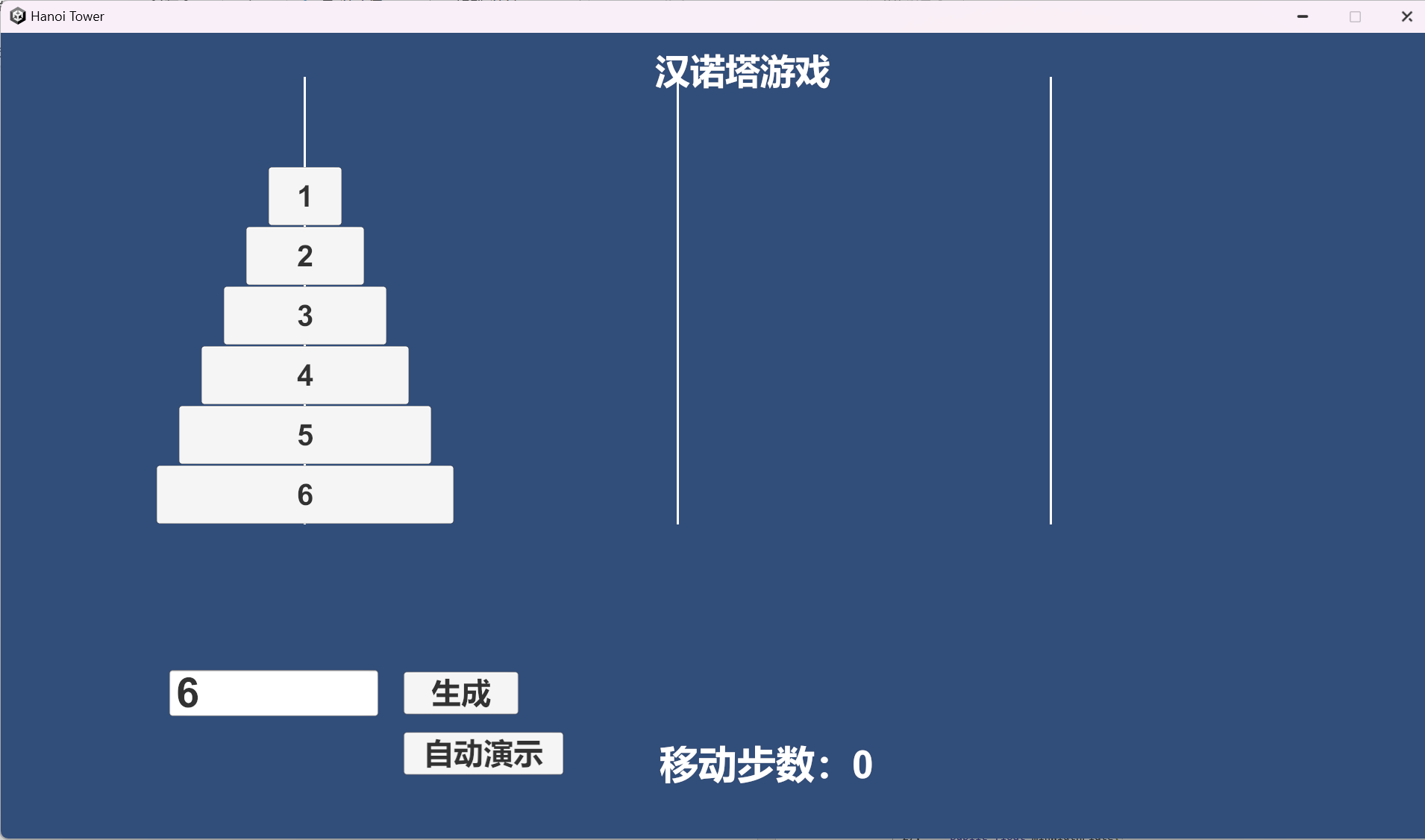
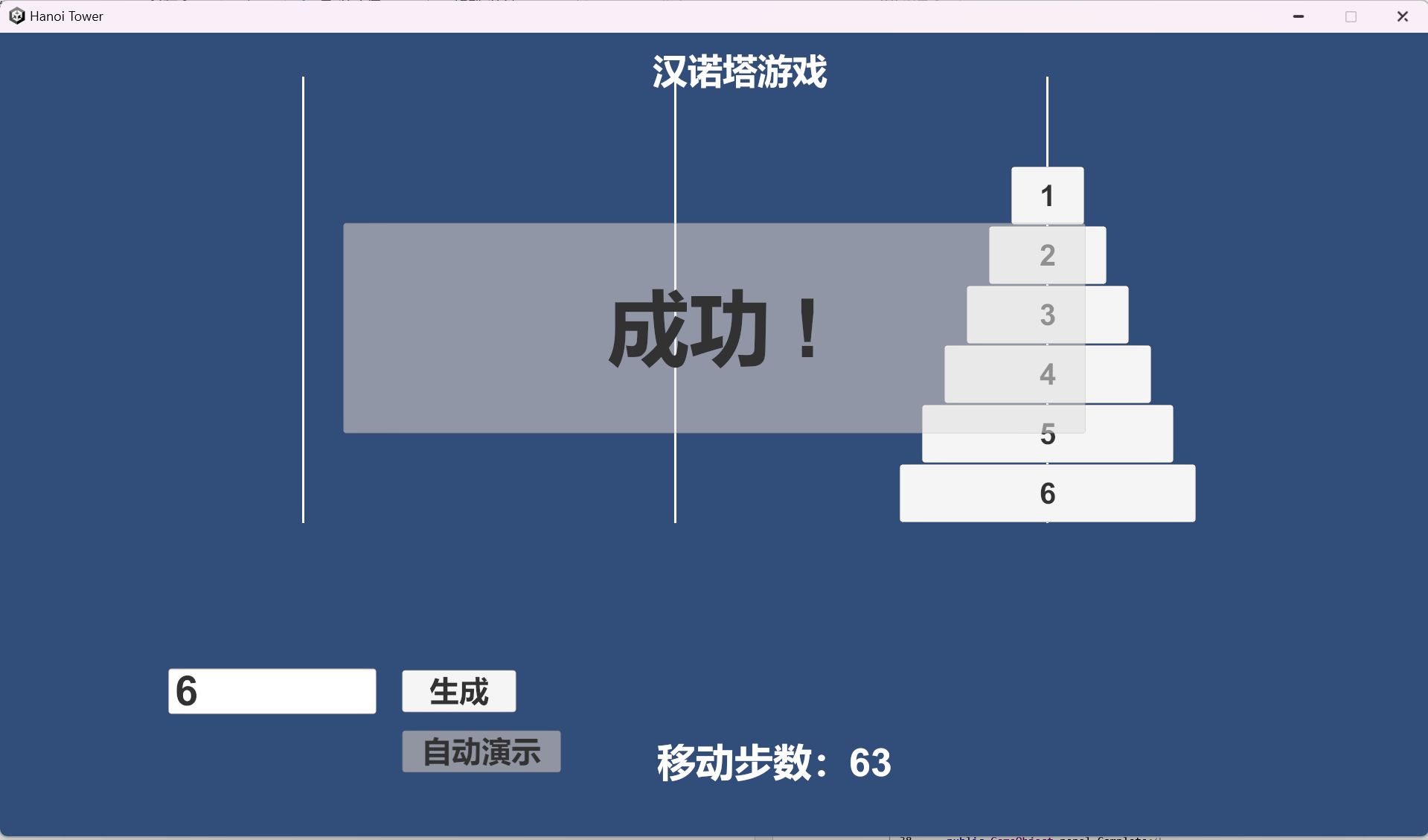
将柱子 b 上的 n-1 个盘子借助柱子 a 移动到柱子 c。

通过递归调用 Execute() 方法，每次将问题规模减小，直到只剩下一个盘子时，就完成了整个汉诺塔的移动过程。n个盘子的最优解的移动步数为。

例如，3个盘子的游戏运行前后的截图如下：

又如，6个盘子的游戏运行前后的截图如下：

**可以在作业同级文件夹找到游戏压缩包，解压后双击运行Hanoi Tower.exe即可。**

附：关键脚本GameManager.cs源码，其中包含关键递归函数Execute()

1. using System.Collections;

2. using System.Collections.Generic;

3. using UnityEngine;

4. using UnityEngine.UI;

5.

6. public class GameManager : MonoBehaviour

7. {

8. public static GameManager instance;

9.

10. private List<GameObject> list\_columns = new();

11. public GameObject prefab\_column;

12. public GameObject panel\_column;

13.

14. [Header("柱子参数")]

15. public float intervalColumn = 300f;

16. public float heightColumn = 750f;

17. public float widthColumn = 30f;

18.

19. [Header("盘子参数")]

20. public int countPlate = 3;

21. public Text text\_countPlate;

22. [HideInInspector]

23. public float heightPlate;

24. [HideInInspector]

25. public float maxWidthPlate;

26. [HideInInspector]

27. public float minWidthPlate;

28.

29. [Header("等待递归函数执行结束的CD")]

30. public float timeWaitForExecute = 1f;

31. [Header("等待递归函数执行结束的计时器,每次移动时刷新计时")]

32. public float timerWaitForExecute;

33. [Header("每次移动的时间间隔")]

34. public float intervalMove = 0.3f;

35. public int maxCountPlate = 20;

36. public bool hasExecuted = false;

37. public Button button\_Generate;

38. public GameObject panel\_Complete;

39. public Text text\_countMove;

40. public GameObject text\_Optimum;

41. public int countMove = 0;

42. public class MoveInfo

43. {

44. public int source;

45. public int destination;

46. public MoveInfo(int s,int d)

47. {

48. source = s;

49. destination = d;

50. }

51. }

52. [SerializeField]

53. List<MoveInfo> list\_moveInfos = new();

54. private void Awake()

55. {

56. instance = this;

57. }

58. void Update()

59. {

60. //if (Input.GetKeyDown(KeyCode.G))

61. // GenerateColumn();

62.

63. //if (Input.GetKeyDown(KeyCode.R))

64. // CallExecute();

65. }

66.

67. public void GenerateColumn()

68. {

69. if (countPlate >= maxCountPlate || countPlate <= 0)

70. return;

71.

72. StopCoroutine(nameof(AutoMove));

73. bool isNum = int.TryParse(text\_countPlate.text, out countPlate);

74. if (!isNum)

75. return;

76. heightPlate = heightColumn / (countPlate + 1.5f);

77. maxWidthPlate = intervalColumn \* 0.8f;

78. minWidthPlate = intervalColumn \* 0.2f;

79.

80. countMove = 0;

81. text\_countMove.text = countMove.ToString();

82.

83. list\_moveInfos.Clear();

84. list\_columns.Clear();

85. ClearChild(panel\_column);

86. float x0 = intervalColumn \* -0.5f;

87. for(int i=0;i<3;i++)

88. {

89. GameObject t\_column = Instantiate(prefab\_column,panel\_column.transform);

90. t\_column.GetComponent<RectTransform>().localPosition = new Vector2(x0 + (i+1)\* intervalColumn, heightColumn/2);

91. t\_column.GetComponent<RectTransform>().sizeDelta = new Vector2(widthColumn, heightColumn);

92. t\_column.SetActive(true);

93. list\_columns.Add(t\_column);

94. }

95. list\_columns[0].GetComponent<Column>().GeneratePlate();

96.

97. button\_Generate.interactable = true;

98. hasExecuted = false;

99. panel\_Complete.SetActive(false);

100. text\_Optimum.SetActive(false);

101. }

102. void move(int x, int y)

103. {

104. timerWaitForExecute = timeWaitForExecute;

105. list\_moveInfos.Add(new(x, y));

106. }

107. public void CallExecute()

108. {

109. if (countPlate >= maxCountPlate)

110. return;

111. if (hasExecuted)

112. return;

113. button\_Generate.interactable = false;

114. hasExecuted = true;

115.

116. list\_moveInfos.Clear();

117. timerWaitForExecute = timeWaitForExecute;

118. WaitForExecute();

119. Execute(countPlate, 0, 1, 2);

120. }

121. void Execute(int n, int a, int b, int c)

122. {

123. if (n == 1)

124. {

125. move(a, c);

126. }

127. else

128. {

129. Execute(n - 1, a, c, b);//将A座上的n-1个盘子借助C座移向B座

130. move(a, c); //将A座上最后一个盘子移向C座

131. Execute(n - 1, b, a, c);//将B座上的n-1个盘子借助A座移向C座

132. }

133. }

134.

135. void WaitForExecute()

136. {

137. if(timerWaitForExecute >= 0f)

138. {

139. timerWaitForExecute -= 0.02f;

140. Invoke(nameof(WaitForExecute),0.02f);

141. return;

142. }

143. //Debug.Log("End WaitForExecute()");

144. StartCoroutine(nameof(AutoMove));

145. }

146. IEnumerator AutoMove()

147. {

148. int c = 0;

149. while(c < 1e5)

150. {

151. c++;

152. if (list\_moveInfos.Count > 0)

153. {

154. countMove++;

155. text\_countMove.text = countMove.ToString();

156.

157. MoveInfo t\_info = list\_moveInfos[0];

158. //Debug.Log("move " + t\_info.source + " -> " + t\_info.destination);

159. list\_columns[t\_info.destination].GetComponent<Column>().PushPlate(list\_columns[t\_info.source].GetComponent<Column>().PopPlate());

160. list\_moveInfos.Remove(t\_info);

161. yield return new WaitForSeconds(intervalMove);

162. }

163. else

164. {

165. panel\_Complete.SetActive(true);

166. break;

167. }

168.

169. }

170. }

171. public bool CheckInit()

172. {

173. //if(list\_columns[0].transform.childCount == countPlate)

174. if(countMove > 0)

175. {

176. button\_Generate.interactable = false;

177. return false;

178. }

179. button\_Generate.interactable = true;

180. return true;

181.

182. }

183. public bool CheckComplete()

184. {

185. if (list\_columns[2].transform.childCount == countPlate)

186. {

187. hasExecuted = true;

188. panel\_Complete.SetActive(true);

189. if (countMove == Mathf.Pow(2, countPlate) - 1)

190. text\_Optimum.SetActive(true);

191. return true;

192. }

193. return false;

194. }

195. public void ClearChild(GameObject p)

196. {

197. for (int i = 0; i < p.transform.childCount; i++)

198. if(p.transform.GetChild(i).gameObject.activeSelf)

199. Destroy(p.transform.GetChild(i).gameObject);

200. }

201. }

202.