웹클라이언트 컴퓨팅
Project Report

숨어있는 펭귄 찾기 게임

20141461
시각디자인
홍영준

1. 게임 기획의도

2. 업그레이드

3. 소스코드 설명 & 상황별 실행화면

1. 게임 기획의도

기본 상황

1) 게임시작버튼, 남은 수 & 실패 수, 시간, 안내 메세지 등을 보여주는 상태표시창 / 게임 진행 창 2가지로 크게 구분

2) 게임 시작 전 숨은 그림을 일정시간 보여줌

3) 일정시간 후 그림을 숨기고, 마우스 클릭으로 찾음

4) 제한 조건에 따라 성공, 실패여부 결정

기획 의도

복잡한 기능을 많이 넣으려고 하기보다

1)게임 진행중 자잘한 오류가 발생하지 않도록 신경 쓸 것과

2)코드의 중복을 최소화하고

3)가능한 다양한 기능, 예를들어 프로토타입 객체, case문, 함수 세분화, document객체의 메소드들을 이용한 html태그 생성, 삽입 등 그 동안 배운 것들을 복습하고, 관련된 다른 개념들을 공부하고자 하였다.

막상 작업을 하다보니 생각지 못한 오류가 발생해서 scope나 node에 관련된 내용도 공부할 수 있게 되었다.

2. 업그레이드

1. 디자인 요소

강아지 -> 펭귄으로 변경하고 클릭 시 fail을 알리는 이미지 추가 제작

2. 업그레이드 개발 요소

1) 레벨 업 버튼으로 알의 갯수를 5, 6, 7의 제곱수로 조절해서 난이도를 선택할 수 있게 함.

2) 게임이 시작되면 5초의 카운트를 표시해줌

3) fail을 알리는 이미지가 1초 보였다가 다시 알의 이미지로 바뀌는 기능 추가

4) 제한시간 외에도 life 개념의 남은 기회를 부여해서 기회를 모두 소진하면 fail

3. 소스코드 설명 && 상황별 실행화면

< index.html >

< style.css >

```
47    .menu-box {
48        width: 100%;
49        border-radius: 20px;
50        margin: 5px 0px 15px 0px;
51        padding: 10px 10px;
52        background: snow;
53
54    }
55
```

1) 나머지 태그는 자바스크립트로 생성하고 40열의 DIV태그의 경우 menu-box 클래스의 css값 중 font-size와 font-weight값을 무시하고 덮어야할 필요가 있어서 태그에 직접 style을 적용하여 우선순위를 높게 두었다. css 파일에서 따로 선택하기보다 효율적이라고 판단했다.

2) 3개의 버튼이 onclick이벤트 시 동일한 함수를 호출하는데, 차이를 주기위해 파라미터 값을 다르게 준다

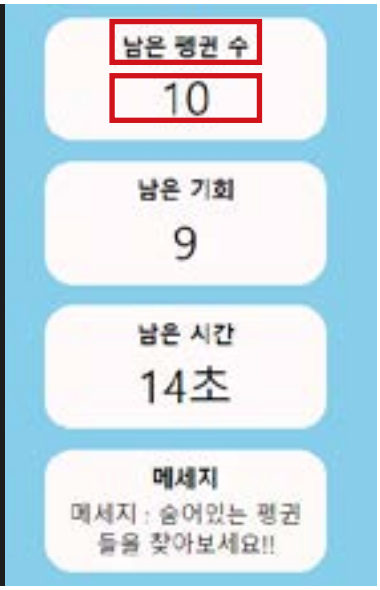
< style.css >

```
13
14    #title {
15        display: flex;
16        justify-content: center;
17        align-items: center;
18        color: #084963;
19        height: 100px;
20        font-size: 1.2em;
21    }
22
```

1) display block인 자식요소를 수평정렬하기 위해 부모태그에 display : flex 적용

< style.css >

```
56    .menu-box :first-child {
57        font-weight: 600;
58    }
59
60    .menu-box :last-child {
61        font-size: 2em;
62        margin-top: 4px;
63    }
64
```



1) menu-box 클래스의 하위요소 2개가지를 각각 선택하기 위해 가상클래스 선택자 사용

< style.css >

```
65  #screen {
66      position: relative;
67      height: 100%;
68      margin-left: 40px;
69  }
```

```
97  #gameover {
98      font-size: 150px;
99      font-weight: 800;
100     position: absolute;
101     top: 50%;
102     left: 50%;
103     transform: translate(-50%,
```

< game.js >

```
function failGame() {
    for (let i = 0; i < answer_card.length; i++) {
        tds[i].firstChild.src = answer_card[i]

    }
    isPlaying = false
    text_message.innerText = '실패! 다시 도전해보세요!'
    gameover.innerText = 'GAME OVER'
}
```

1) 카운트 다운을 이미지 위에 겹쳐보이게 하기위해 position absolute사용. 그 부모요소를 position relative 적용

2) javascript로 게임오버시 호출되는 함수에 해당 태그의 innerText를 변경하여 띄움

< game.js >

```
let count_fail = 0
let count_remain = 0

const text_remain = document.getElementById('remain')
const text_fail = document.getElementById('fail')
const text_time = document.getElementById('time')
const text_message = document.getElementById('message')
const btn_level_1 = document.getElementById('btn_level_1')
const btn_level_2 = document.getElementById('btn_level_2')
const btn_level_3 = document.getElementById('btn_level_3')
const screen = document.getElementById('screen')

const img_egg = 'img/img_egg.png'
const img_penguin = 'img/img_penguin.png'
const img_fail = 'img/img_fail.png'

let isPlaying = false
let tds = []
let answer_card = []
```

1) 전역변수 선언부.

각종 html dom 객체들과 자주 사용될 이미지 src, 게임이 진행중인지 구분하기 위한 isPlaying, 현재 게임안에 보이는 이미지 태그의 부모요소를 모은 배열 tds, 정답 카드뭉치 배열 answer_card

< index.html >

```
<body onload=set()>
    <div id="title">
        <img src="./img/img_fail.png" alt="egg"
        <img src="./img/img_egg.png" alt="egg">
```

< game.js >

```
function set() {
    createTable(5)
    insertImg(5)

}
```

1) body가 onload되면 처음으로 불리는 js함수. 세팅을 위해 두 함수를 호출한다.

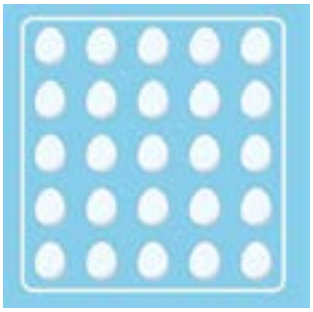
< game.js > - createTable(level)

```
27
28  function createTable(level) {
29      while (screen.hasChildNodes()) {
30          screen.removeChild(screen.firstChild)
31      }
32
33      const table = document.createElement('table')
34      screen.appendChild(table)
35
36      for (let i = 0; i < level; i++) {
37          const tr = document.createElement('tr')
38          table.appendChild(tr)
39          for (let j = 0; j < level; j++) {
40              const td = document.createElement('td')
41              tr.appendChild(td)
42          }
43      }
44      const gameover = document.createElement('div')
45      gameover.id = 'gameover'
46      screen.appendChild(gameover)
47
48  }
49
```

1) 호출시 최초에 while문에서 게임 진행창인 screen의 자식노드가 있는지 hasChildNodes()로 확인하면서 모든 자식 요소를 삭제한다

2) 그 후 새로운 table태그를 생성, 파라미터값에 따라 이중 반복문을 통해 tr, td 태그를 생성해 게임 이미지가 들어갈 틀을 생성한다.

3) 모든 table이 생성된 후 gameover 이미지를 띄울 빈 div태그를 생성해둔다. table 작업이 끝난 후에 생성해서 추가해야하기 때문에 선언도 이곳에서 한다. 이하 함수들은 모두 이 함수가 호출된 후에 각각 호출되기 때문에 여기서 const로 선언해도 scope에 문제가 생기지 않는다. 오히려 set()이나 전역변수로 선언해두면 문제가 생긴다.

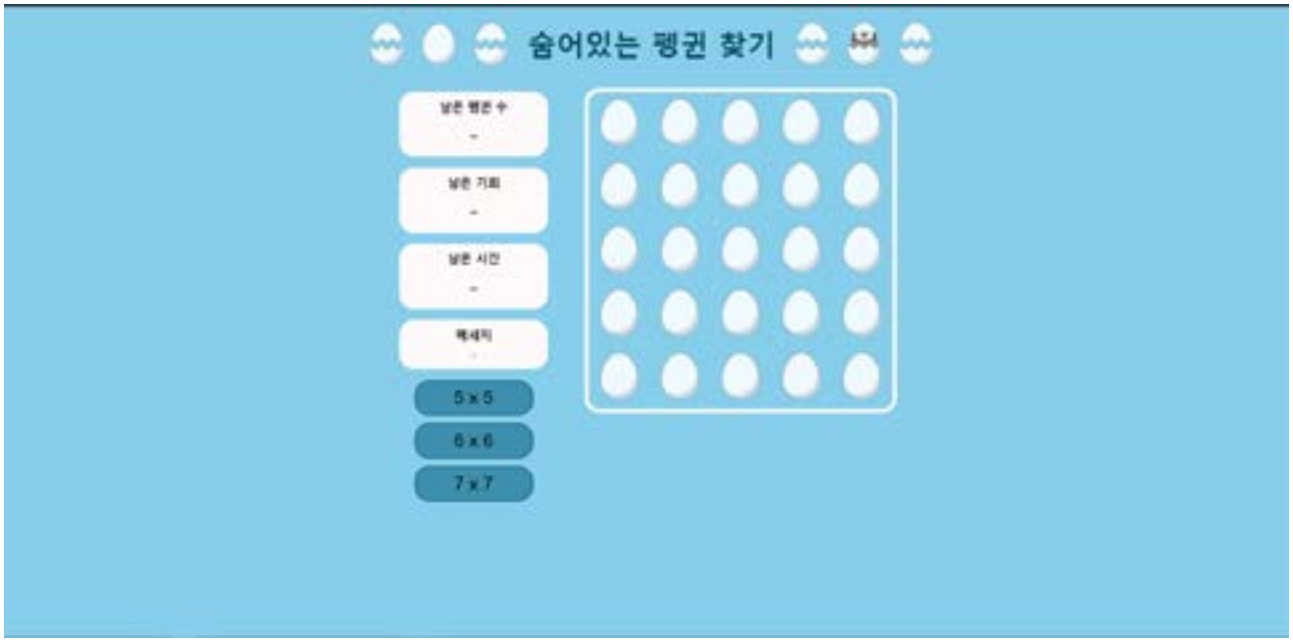


< game.js > - insertImg(level)

```
50  function insertImg(level) {
51      let i = 0
52      tds = screen.getElementsByTagName('td')
53      while (i < level * level) {
54          const img = new Image()
55          img.setAttribute('src', img_egg)
56          img.setAttribute('width', '80px')
57          tds[i].appendChild(img)
58          i++
59      }
60  }
61
```

1) createTable(level)이 생성한 table의 td들에 img 객체를 생성, 삽입하는 함수이다. 굳이 createTable(level)안에 넣지않고 따로 함수로 구분한 이유는 두 가지.

첫 번째는 해당 코드가 어떤 동작을 하는지 명시하기 위함,
두 번째는 이후에 설명할 gameStart(level)함수가 createTable(level)를 호출하는데 이때 insertImg(level)의 내용이 포함되어 있으면 오류는 나지 않지만 불필요한 img 객체 생성, 삽입의 동작이 반복되기 때문에 제거하기 위함이다.



set()함수가 createTable(level), insertImg(level)을 호출하여 생성한 첫 화면

< game.js > - reset()

```
61
62  function reset() {
63      isPlaying = false
64      count_fail = 10
65      count_remain = 0
66      answer_card = []
67      text_remain.innerText = '-'
68      text_fail.innerText = '-'
69      text_time.innerText = '-'
70      text_message.innerText = '-'
71  }
72
```

1) 게임이 다시 시작될 때 마다 리셋되어야할 정보를 다시 초기값으로 되돌리는 함수.

< game.js > - gameStart(level)

```
function gameStart(level) {
    createTable(level)
    reset()
    text_message.innerText = '메세지 : 펭귄들의 위치를 기억하세요!!'
    btn_level_1.style.visibility = "hidden"
    btn_level_2.style.visibility = "hidden"
    btn_level_3.style.visibility = "hidden"
    tds = screen.getElementsByTagName('td')
    let count = 5000
    for (let i = 0; i < level * level; i++) {
        const card = new Card()
        card.img.onclick = card.click
        card.img.setAttribute("width", '80px')
        if (card.isPenguin) {
            answer_card.push(img_penguin)
        } else {
            answer_card.push(img_egg)
        }
        tds[i].appendChild(card.img)
    }
    countDown()
    setTimeout(function () {
        for (let i = 0; i < tds.length; i++) {
            tds[i].firstChild.setAttribute('src', img_egg)
        }
        setTime()
        isPlaying = true
        btn_level_1.style.visibility = "visible"
        btn_level_2.style.visibility = "visible"
        btn_level_3.style.visibility = "visible"
        text_fail.innerText = count_fail
        text_remain.innerText = count_remain
    }, count)

    count_fail = 10

}
```

1) createTable과 reset으로 게임 환경을 초기화하고 게임 진행을 위해 전달받은 level파라미터에 따라 for문을 통해 Card() 프로토타입으로 객체를 생성하여 테이블에 삽입한다. 이 과정에서 card객체의 속성에 따라 answer_card에 정답 정보를 담는다.

2) 시작과 동시에 countDown()을 호출해서 카운트다운을 진행하고 그만큼 setTimeout으로 코드 실행을 지연시켜 setTime()으로 게임 시간을 설정한다.

3) 상태창을 적절히 변형시켜 현재상황을 안내한다.

< game.js > - Card()

```
109
110  function Card() {
111
112      this.isPenguin = Math.random() >= 0.5
113      const isPenguin = this.isPenguin
114      let getAnswer = false
115      if (this.isPenguin) {
116          count_remain++
117      }
118
119      this.img = new Image();
120
121      const img1 = new Image();
122      img1.src = img_egg
123      const img2 = new Image();
124      img2.src = img_penguin
125
126      if (this.isPenguin) {
127          this.img = img2
128      } else {
129          this.img = img1
130      }
131
132      this.click = function () {
133          if (isPlaying) {
134              const self = this
```

1) 각 카드의 프로토타입으로 this.isPenguin 함수에 50%확률로 true, false를 할당한다. 또한 this.getAnswer로 현재 이 카드가 정답이고, 유저가 맞춘 상태인지 아닌지 구분해서 중복 클릭이 되지 않도록 방지한다.

2) click 함수를 내장하고 있다.

3) const isPenguin = this.isPenguin 으로 재할당하는 이유는 내포하고 있는 this.click()함수가 이 정보를 사용해야하기 때문이다.

< 5x5 >

< 6x6 >

< 7x7 >

< game.js > - this.click()

```
132     this.click = function () {
133         if (isPlaying) {
134             const self = this
135             if (isPenguin && !getAnswer) {
136                 this.setAttribute('src', img_penguin)
137                 count_remain--
138                 getAnswer = true
139                 if (count_remain <= 0) {
140                     text_remain.innerText = count_remain
141                     winGame()
142 
143                 }
144             } else if (isPenguin && getAnswer) {
145                 return
146             } else {
147                 this.setAttribute('src', img_fail)
148 
149                 setTimeout(function () {
150                     self.setAttribute('src', img_egg)
151                 }, 1000)
152 
153                 count_fail--
154 
155                 if (count_fail < 1) {
156                     failGame()
157 
158                 }
159             }
160             text_remain.innerText = count_remain
161             text_fail.innerText = count_fail
162         }
163     }
164 }
165
```

1) 먼저 isPlaying으로 게임 진행중일 때만 클릭이 가능하도록 제한을 걸고 시작한다.

2) conse self = this로 호출한 이미지 객체를 self 변수에 저장하는 이유는 위의 isPenguin과 마찬가지로, line149의 setTimeout함수에서 this는 최초 호출한 이미지 객체가 아닌 그 함수를 호출한 click함수가 되기 때문에 이미지 객체를 지칭하기 위해서이다.

3) 클릭한 이미지 객체의 isPenguin값에 따라 클릭했을 때 동작을 달리하고, 적절한 이미지로 변경, 값 수정등이 실행된다.

4) 남은 기회를 모두 소진하면 게임 실패를 알리는 failGame()을 호출한다

< game.js > - countDown(), setTime()

```
166  function countDown() {
167      const count_num = document.getElementById('count')
168      let time = 5
169      setTimeout(function count() {
170          if (time > 0) {
171              count_num.innerText = time
172              time--
173              setTimeout(count, 1000)
174          } else {
175              count_num.innerText = ''
176
177          }
178      })
179  }
180
181  function setTime() {
182      text_message.innerText = '메세지 : 숨어있는 동전들을 찾아보세요!!'
183
184      let time = 20
185      setTimeout(function count() {
186          if (time > 0 && isPlaying) {
187              text_time.innerText = time + '초'
188              time--
189              setTimeout(count, 1000)
190          } else if (time <= 0) {
191              text_time.innerText = time + '초'
192              failGame()
193          }
194      })
195  }
196
```

1) 시간과 관련된 함수로 게임실행시 카운트다운 5초를 실행하는 countDown()과 재귀를 통해 1초에 한 번씩 스스로를 호출하며 게임의 남은시간을 나타내는 setTime()함수이다. 재귀의 경우 시간이 0이되면 멈추고, 게임의 패배를 알리는 failGame()을 호출한다.

conutDown()

setTime()

< game.js > - failGame(), winGame()

```
198
199  function failGame() {
200      for (let i = 0; i < answer_card.length; i++) {
201          tds[i].firstChild.src = answer_card[i]
202
203      }
204      isPlaying = false
205      text_message.innerText = '실패! 다시 도전해보세요!'
206      gameover.innerText = 'GAME OVER'
207  }
208
209  function winGame() {
210      isPlaying = false
211      const lev = tds.length
212      switch (lev) {
213          case 25:
214              text_message.innerText = '1단계 성공! 다음 단계에 도전하세요!!'
215              break
216          case 36:
217              text_message.innerText = '2단계 성공! 다음 단계에 도전하세요!!'
218              break
219          case 49:
220              text_message.innerText = '최고 단계를 성공하셨습니다!!'
221              break
222      }
223      gameover.innerText = 'SUCCESS'
224  }
```

1) 게임 성공, 실패 조건 만족시 호출되어 게임 진행을 멈추고 성공의 경우 성공 메세지와 함께 다음 단계 안내를, 실패의 경우 정답카드를 보여주고 재도전을 촉구하는 안내 메세지를 보여준다.