Архітектура програмного забезпечення

Лабораторна робота N3

Тема: Цикл подій (event loop)
Мета: Отримання навичок імплементації систем на основі циклу подій. Знайомство з його використанням у UI фреймворках.

Завдання

Робота складається з 3-х частин:

1. Імплементація мінімального графічного інтерфейсу користувача за допомогою циклу подій

2. Реалізацію окремого “циклу подій” для обробки зовнішніх команд для контролю зображення, яке відмальовується попереднім компонентом.

3. Фіналізація: побудова діаграми залежностей між компонентами вашої системи, CI, скрипт для зовнішніх команд.

Частина 1: графічний інтерфейс

На базі шаблону, який ви можете знайти у репозиторії за посиланням нижче

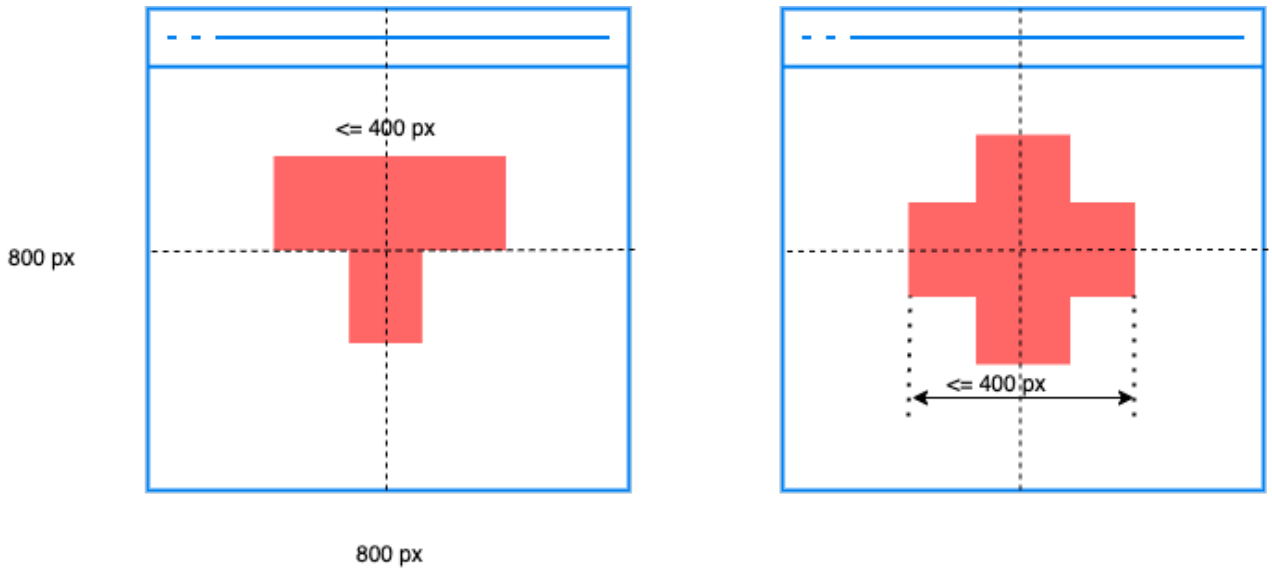
https://github.com/roman-mazur/architecture-lab-3

вам потрібно реалізувати відображення вікна, у якому промальовується фон кольору, який вказаний у варіанті та фігура вашого варіанта.

- Фігура має бути розміщена по центру вікна, її розміри мають бути не більшими за половину розміру вікна. Фінальні пропорції ви можете підібрати на власний смак.

- Вікно має бути “квадратним”: 800x800 px

- Схематичне зображення фігур “T” та “хрестик” зображено нижче. Обидві фігури можуть бути отримані за допомогою малювання прямокутників.



Для цієї реалізації, вам, в основному, потрібно реалізувати обробку події `paint.Event` у `ui.Visualizer`. Для більших деталей про роботу циклу подій, звертайтеся до відповідних матеріалів на Classroom.

Прямокутник можливо намалювати за допомогою методу Fill типу screen.Window. Перший аргумент вказує позицію та розмір прямокутника.

Після імплементації відображення фону (малюючи прямокутники) вам також потрібно додати інтерактивний елемент: реагувати на натискання кнопки миші та змінювати позицію фігури, переміщуючи її центр в координати миші. Залежно від варіанта вам потрібно реагувати або на ліву кнопку миші, або на праву.

Перевірку роботи вашого інтерфейсу ви можете виконувати запускаючи команду в `cmd/painter`.

Частина 2: цикл подій для опрацювання скрипта малювання

У цій частині вам потрібно реалізувати інтерпретатор команд, який виконує інструкції у циклі подій, оновлюючи текстуру (буфер із зображенням) та передаючи її у `ui.Visualizer` для відображення на екрані. Інтерпретатор повинен читати команди з тіла HTTP запита. Кожна команда займає один рядок та має формат

`<`**`інструкція`**`> [<`**`аргумент`**`> ...]`

Різні інструкції можуть вимагати різну кількість аргументів (включно з 0). Інструкція та аргументи розділені хоча б одним символом пропуску.

У даній роботі потрібно буде реалізувати обробку наступних інструкцій:

1. **white**
   Зафарбувати фон білим кольором

2. **green**
   Зафарбувати фон зеленим кольором

3. **update**
   Передати сформовану текстуру у ui.Visualizer для відображення у вікні графічного інтерфейсу

4. **bgrect <x1> <y1> <x2> <y2>**
   Намалювати на фоні прямокутник чорного кольору у вказаних координатах. Про формат координат читайте нижче. На фоні може бути лише один такий прямокутник. Якщо ця команда зустрічається кілька разів, то видимим на фінальному зображенні буде лише прямокутник з останніми координатами.

5. **figure <x> <y>**
   Намалювати нову фігуру вашого варіанта з центром у вказаних координатах поверх сформованого фону.

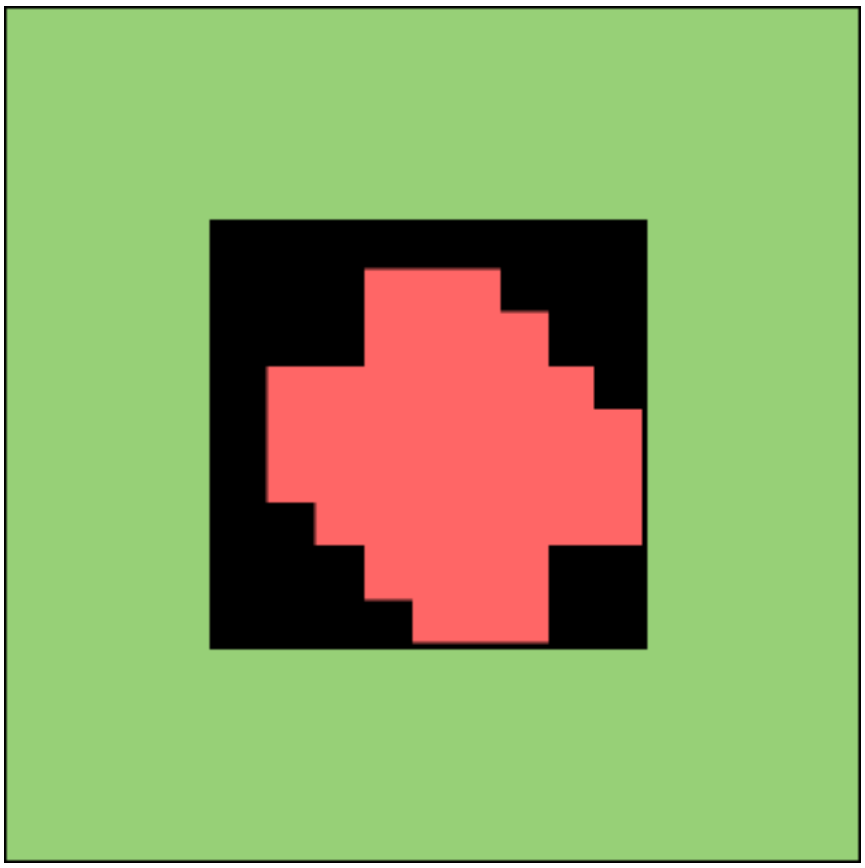
6. **move <x> <y>**
   Перемістити усі фігури попередньо намальовані за допомогою команди figure у вказані координати.

7. **reset**
   Очистити весь поточний стан текстури (інформацію про колір фону, чорний прямокутник, усі фігури додані через команду figure), зафарбувати її чорним кольором (залишити лише фон з чорним кольором).

Зауважте, що з опису команда вам потрібно буде окремо працювати з оновленням фону та фігур поверх нього: вам доведеться тримати стан, який описує ці два компонента. Наступний скрипт

```
white

bgrect 0.25 0.25 0.75 0.75

figure 0.5 0.5

green

figure 0.6 0.6

update
```

повинен відобразити вікно з фоном у вигляді чорного квардата з жирною зеленою рамкою та двома фігурами на цьому фоні, які накладаються оду на одну. Приклад для червоної фігури “хрестик” нижче.



Обробник HTTP запиту має читати тіло запиту (поле Body структури http.Request) рядок за рядком, парсити команду в прочитаному рядку та додавати команду в чергу обробки циклу подій. Для задачі парсинга найзручніше скористатися *Scanner з пакету bufio*.

Код парсера (у `painter/lang/parser.go`) може мати приблизно наступний вигляд:

```
  scanner := bufio.NewScanner(in)
  scanner.Split(buffo.ScanLines)

  var res []painter.Operation

  for scanner.Scan() {
    commandLine := scanner.Text()
    op := parse(commandLine) // parse the line to get Operation
    res = append(res, op)
  }

  return res
```

Для реалізації функції parse вам також може знадобитися функція `strings.Fields`.

Перед тим як писати код для 2-гої частини завдання, зверніть увагу, що якщо ви запустите `cmd/painter` та відкриєте наступне посилання у своєму браузері, то повинні побачити, що вікно вашої програми зафарбовується у білий колір.

http://localhost:17000/

Це відбувається через те, що у програмі вже зв'язано кілька компонентів:

- Коли `ui.Visualizer` ініціалізує UI toolkit (shiny) та отримує `screen.Screen`, запускається `painter.Loop`, який ще не вміє працювати з чергою повідомлень, але просто виконує отримані команди, коли вони йому передаються в методі `Post` та вміє передавати готову текстуру своєму `painter.Receiver` (яким виступає `ui.Visualizer`).

- На початку програми також запускається HTTP сервер, який реагує на вхідні запити, віддає тіло запиту у `lang.Parser`, а результат парсингу передає у `painter.Loop` через метод `Post`.

- Реалізація lang.Parser поводиться так, ніби на вхід їй прийшов скрипт нижче (ігноруючи реальні дані):

```
white
update
```

Для виконання 2-ої частини завдання вам потрібно зробити наступне.

- Додати реалізацію черги повідомлень та власне циклу подій у `painter.Loop`

- Реалізувати справжній парсер (`lang.Parser`) для команд описаних вище.

- Додати імплементації `painter.Operation` для цих команд з урахуванням необхіності підтримки стану фону та фігур.

- Переконатися, що черга повідомлень, `painter.Loop` та парсер (`lang.Parser`) покриті тестами (без тестів ви втратите бали).
  Тестування `ui.Visualizer` та поведінки конкретних імплементацій `painter.Operation` лишається факультитивним завданням.

Частина 3: фіналізація

У цій роботі ви працюєте з програмою, яка складається з кількох компонентів. Побудуйте діаграму залежностей цих компонентів. Чи отримали ви направлений ациклічний граф? Виконану діаграму потрібно додати в корінь проекту файлом `components.pdf`.

Також, як і в попердній роботі, на репозиторії вашого проекту має бути налаштовано CI за допомогою Github Actions. CI повинен запускати тести для всіх ваших пакетів та збирати команду `cmd/painter`, щою переконатися, що немає проблем з отриманням виконуваного файла.

І останнім завданням є додавання кількох скриптів/програм, які взаємодіють з HTTP сервером та контролюють ваше вікно графічного інтерфейсу. Ці скрипти/програми ви можете додати одним з 3-х шляхів:

1. Як команди на Go, використовуючи http.Client, щоб відправити скрипт малювання на сервер cmd/painter.

2. Як Bash/Shell скрипти з використанням curl.
3. Як HTML сторінку з JavaScript/TypeScript кодом для взаємодії з HTTP сервером у cmd/painter.

Вам потрібно покрити принаймні 2 сценарія.

1. Відправка команд, які створюють зелену рамку (дивися приклад у частині 2).
2. Малювання фігури вашого варіанта на будь-якому фоні та відправка

   ```
   move <x> <y>
   update
   ```

   кожну секунду для зміни положення фігури на зображенні у вікні, наприклад, рухаючи фігуру по діагоналі.

Інші сценарії вітаються. Скрипти зі сценаріями мають розміщуватися у директорії `scripts` в корені проекта.

Варіанти

Варіант	Фігура	Колір фону	Колір фігури	Клопка миші
1	T	білий	жовтий	ліва
2	Хрестик	білий	жовтий	ліва
3	T	чорний	жовтий	ліва
4	Хрестик	чорний	жовтий	ліва
5	T	зелений	жовтий	ліва
6	Хрестик	зелений	жовтий	ліва
7	T	білий	червоний	ліва
8	Хрестик	білий	червоний	ліва
9	T	чорний	червоний	ліва
10	Хрестик	чорний	червоний	ліва
11	T	зелений	червоний	ліва
12	Хрестик	зелений	червоний	ліва
13	T	білий	синій	ліва
14	Хрестик	білий	синій	ліва
15	T	чорний	синій	ліва
16	Хрестик	чорний	синій	ліва
17	T	зелений	синій	ліва
18	Хрестик	зелений	синій	ліва

Варіант	Фігура	Колір фону	Колір фігури	Клопка миші
19	T	білий	жовтий	права
20	Хрестик	білий	жовтий	права
21	T	чорний	жовтий	права
22	Хрестик	чорний	жовтий	права
23	T	зелений	жовтий	права
24	Хрестик	зелений	жовтий	права
25	T	білий	червоний	права
26	Хрестик	білий	червоний	права
27	T	чорний	червоний	права
28	Хрестик	чорний	червоний	права
29	T	зелений	червоний	права
30	Хрестик	зелений	червоний	права
31	T	білий	синій	права
32	Хрестик	білий	синій	права
33	T	чорний	синій	права
34	Хрестик	чорний	синій	права
35	T	зелений	синій	права
36	Хрестик	зелений	синій	права

Визначення свого варіанта

Для визначення свого варіанта, скористайтеся наступною командою в чаті:

```
/team-variant 3
```

Дана інструкція поверне варіант 3-ої лабораторної роботи для вашої команди.

Оцінювання

Ви можете здобути до 10 балів за дану лабораторну роботу (2, 5 та 3 бала за відповідні частини). Необхідною умовою здачі роботи є імплементація обробки вашого варіанта операцій. У залежності від того, на скільки ви дотримуєтеся архітектури описаної в завданні та деталей імплеменетації, кількість балів буде змінюватися.

Успіхів!