Zápočtová práce z předmětu
Java (NPRG013)
MFF UK

Colos

Uživatelská dokumentace

Autor:
Richard Jedlička

27. května 2011

Obsah

1 Úvod **2**

2 Sestavení **2**
2.1 Stažení 2
2.2 Závislosti 2
2.3 Kompilace 2
2.3.1 Sestavení přímo přes CMake 3
2.3.2 Sestavení pomocí připravených skriptů 3
2.3.3 Po sestavení 3

3 Používání **3**
3.1 Používání základní knihovny 3
3.1.1 keyboard.hpp 4
3.1.2 mouse.hpp 4
3.1.3 system.hpp 5
3.2 Používání rozšíření *Actions* 5

1 Úvod

Colos je aplikace určená pro výběr barvy. Umožnuje vabírat barvu v několika barevných modelech - RGB, HSL, HSV.

2 Používání

2.1 Používání základní knihovny

Pro použití knihovny ve Vaší aplikaci stačí na`inkludovat` potřebné hlavičkové soubory a při kompilaci knihovnu přilinkovat.

Dále bude popsána stručně základní práce s jednotlivými hlavičkovými soubory. Konkrétní detaily lze nalézt v API dokumentaci, kterou lze vygenerovat při sestavování. Případně lze vše dohledat přímo v kódu knihovny.

2.1.1 keyboard.hpp

Tento soubor obsahuje třídů `Keyboard` simulující klávesnici jako zařízení. Je to pouze statická třída, takže se nevytvářejí zádné její instance. Obsahuje metody pro stisk a uvolnění klávesy. Metody přebírají jako parametr objekt třídy `Key`, což je objekt reprezentující konkrétní klávesu. Tento objekt lze získat několika způsoby. Nezávisle na platformě lze instanci třídy `Key` zkonstruovat pomocí tzv. `KeyType` což je výčtový typ obsahující nejběžnější typy kláves anglické klávesnice. Všechny typy lze nelézt v souboru `types.hpp`.

> Objekt reprezentující klávesu `A`, vytvoříme takto:
> `FakeInput::Key keyA(FakeInput::Key_A);`

Pokud potřebujeme klávesu, která není mezi `KeyType`, lze na jednotlivých platformách využít nativních reprezentací typy kláves. Na *Unixu* je to `KeySym` a na *Windows* `Virtual-Key code`. Případně lze „vytáhnout“ typ klávesy z reálné události. Jak konkrétní metody používat naleznete v API dokumentaci.

> Stisk vytvořené klávesy `A` tedy simulujeme takto:
> `FakeInput::Keyboard::pressKey(keyA);`
>
> Později je zase potřeba stisknutou klávesu uvolnit:
> `FakeInput::Keyboard::releaseKey(keyA);`

2.1.2 mouse.hpp

Tento soubor obsahuje třídu `Mouse` simulující myš jako zařízení. Opět je pouze statická. Obsahuje metody pro ovládání tlačítek myši, pro otáčení kolečkem a pro pohym kurzorem. Stisky tlačítek jsou v podstatě stejné jako stisky kláves u klávesnice, akorát se nepracuje s tlačítkem jako s objektem, ale přímo s typem tlačítka (`MouseButton` - opět lze nalézt v souboru `types.hpp`).

```
FakeInput::Mouse::pressButton(FakeInput::Mouse_Left);
FakeInput::Mouse::releaseButton(FakeInput::Mouse_Left);
```

Kolečkem lze točit buď nahoru nebo dolů. Podle toho tedy vybereme buď metodu `wheelUp()` nebo `wheelDown()`.

Pro pohyb s kurzorem lze využít jednu ze dvou metod. A to buď `move(int dx, int dy)` nebo `moveTo(int x, int y)`. Metody se liší v tom, že první zmiňovaná posouvá kurzorem relativně vůči aktuální pozici, zatímco druhá metoda umistujě kurzor na konkrétní pozici na obrazovce.

2.1.3 system.hpp

V tomto souboru naleznete třídu `System`. Ta nesimuluje operační systém nebo něco podobného jak by někomo mohlo z názvu napadnout, nebo ne alespoň celý :-). Obsahuje metody systémového charakteru. Konkrétně dvě, a to `runCommand(const std::string& cmd)` a `wait(unsigned int millisec)` první metoda je v této třídě nejdůležitější a slouží ke spouštění příkazů pro příkazovou řádku. Jako parametr přebírá textový řetězec obsahující příkaz.

> Např. pokud je v proměnné prostředí PATH nastavena cesta ke spouštěcímu soboru prohlížeče Firefox, lze ho spustit jednoduše takto: `FakeInput::System::runCommand("firefox");`

Druhá metoda pouze uspí aktuální vlákno na určený čas v milisekundách. Využití bude popsáno dále.

2.2 Používání rozšíření *Actions*

Součástí knihovny je i její rozšíření *Actions*. Toto rozšíření v podstatě zaobaluje jednotlivá volání metod zmiňovaná výše do objektů. Zatímco v základní části knihovny pracujete se simulací vstupních zařízení, v *Actions* pracujete s

akcemi jako takovými, může te uchovávat v proměnných a provádět když je potřeba. Objekt akce je instance některé ze tříd odvozených od třídy `Action`, což je abstraktní třída předepisující metodu `send()` sloužící pro provedení (odeslání do systému) akce. Potřebné hlavičkové soubory naleznete ve složce `src/actions`.

Pro každé jednotlivé možnosti použití základní části knihovny je zde jedna akce. Tedy např. pro stisk klávesy je tu `KeyboardPress` akce, pro realtivní posun kurzoru myši je tu `MouseRelativeMotion` akce. Výpis všech akcí i s popisi jejich rozhraní naleznete opět v API dokumentaci.

Tou nejzábavnější věcí ale na *Actions* rozšíření je jedna akce, která nemá analogii v základní části knihovny a to je `ActionSequence`. `ActionSequence` slouží k tomu, že lze libovolné akce spojit do řetezce a podle potřeby je najednou v daném pořadí provést. `ActionSequence` je tedy také akcí, ač to na první podled nevypadá. Je potomek třídy `Action`, tzn. implementuje metodu `send()`. Sekvence akcí se celkem snadno používá. Jednoduše vytvoříte její instanci a pak naní voláte metody odpovídající jiným akcím, čímž je zapojíte do řetězce. Zde také konečně přichází na řadu metoda `wait` ze třídy `System`, využije se pokud je třeba udělat mezi některými akcemi v řetežci prodlevu. Například počkat až naběhne aplikace přijímající textový vstup a teprve pak začít „psát“.

```
FakeInput::ActionSequence ac;
ac.runCommand("xterm").wait(500).press(keyA).release(keyA);
ac.send();
```

Sekvence akcí lze i vzájemně spojovat. Tím vznikne řezetec akcí obsahující nejprve akce z první sekvence a hned za nimi akce se sekvence druhé. Ve skutečnosti se druhá připojovaná sekvence v první zařadí na konec jako jakákoli jiná akce, přecijen je sekvence akcí také akce jak bylo zmíňeno na začátku této sekce.

```
FakeInput::ActionSequence ac1;
ac1.moveMouse(100, 0).press(FakeInput::Mouse_Right);
FakeInput::ActionSequence ac2;
ac2.moveMouse(0, 50).press(FakeInput::Mouse_Left);
ac1.join(ac2).send(); // připojí ac2 k ac1 a hned provede
```