Институт за математику и информатику
Природно-математички факултет
Универзитет у Крагујевцу

Семинарски рад из предмета Логичко и функцијско програмирање

Cycpick

Ментор:
dr Татјана Стојановић

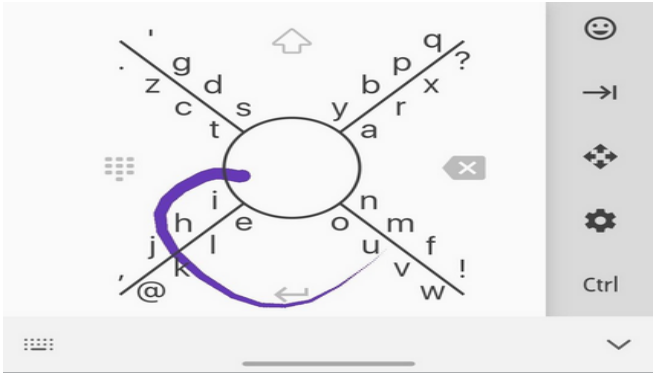
Студент:
Срђан Тодоровић 87/2019

Садржај

Увод

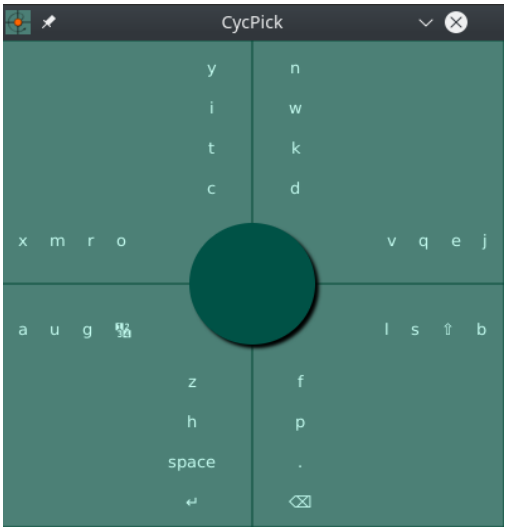
Пројекат *CycPick* има за циљ да истражи и прикаже могуће начине за побољшање интеракције између људи и рачунара које се могу реализовати коришћењем спољашњег уређаја рачунара који се назива миш.

Инспирација за овај пројекат потиче из идеје пројекта *8Vim*, мобилне апликације која представља уређивач текста чији је главни фокус повећање ефикасности уређивања, сличан програму *Vim*, али за мобилне уређаје. Глава идеја апликације *8Vim* јесте одабир опција кроз вршење цикличних покрета који су слични природним покретима који се врше приликом ручног писања текста. Као комбинација ове идеје и идеје програма *Vim* настала је идеја пројекта *CycPick*, коришћење цикличних покрета за ефикасан одабир одређене опције из скупа опција.



Слика 1: Одабир слова "e" помоћу апликације 8Vim

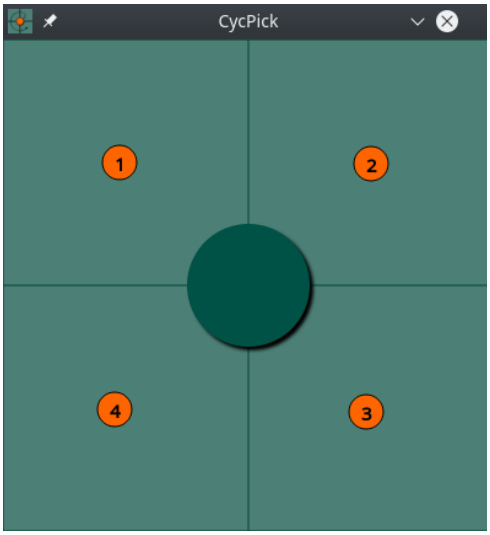
У програму *CycPick* корисник врши покрете користећи рачунарски миш. Његовим пролазком кроз одређену ивицу једног од четири квадрата бира групу опција и тиме сужава скуп опција које се посматрају. Након одабира групе корисник може одабрати опцију коју је наумио да одабере. Детаљнији опис функционисања програма биће описан у наредним одељцима.



Слика 2: Изглед програма CycPick

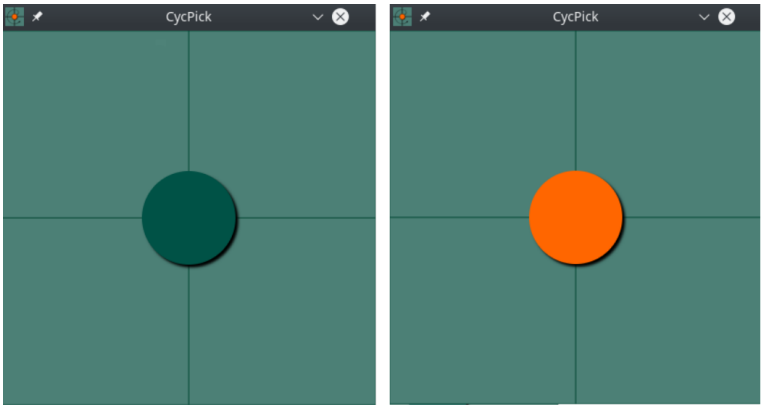
Опис изгледа и рада програма

Главни приказ програма је подељен на четири блока (квадрата) који су нумерисани у смеру окретања казаљке на сату почевши од горњег левог блока (Слика 3). Осим блокова, главни приказ чини и дугме кружног облика које се налази у његовом централном делу.



Слика 3: Приказ нумерације блокова

Централно дугме представља почетну и завршну тачку сваког потеза, односно одабира. Уколико фаза бирања није у току ово дугме је тамно зелено боје, а уколико јесте у том случају ће бити обојено наранџастом бојом (Слика 4).



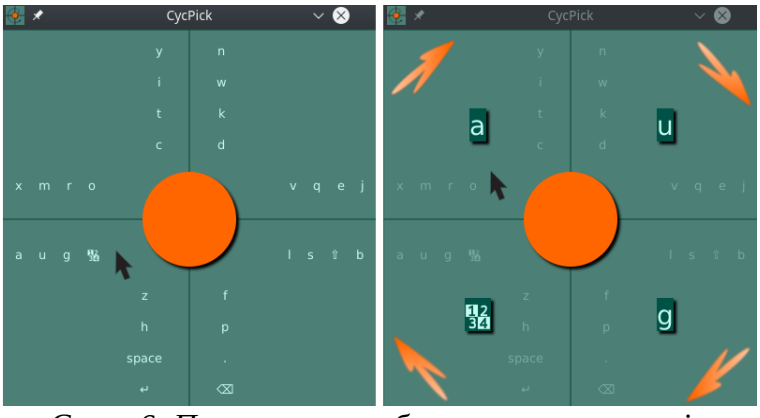
Слика 4: Прикази централног дугмента у случајевима када фаза бирања није (лево) и када јесте (десно) у току

Последња компонента која чини главни приказ програма јесте скуп група које садрже опције. На заједничким ивицама блокова налазе се групе. Сваки блок садржи по две групе које се налазе унутар њега, по једну за сваку ивицу коју дели са другим блоком. Групе су, као и блокови, нумерисане у смеру окретања казаљке на сату, где хоризонтална група горњег левог блока представља прву групу.

Слика 5: Приказ нумерације група опција

Процес бирања опције

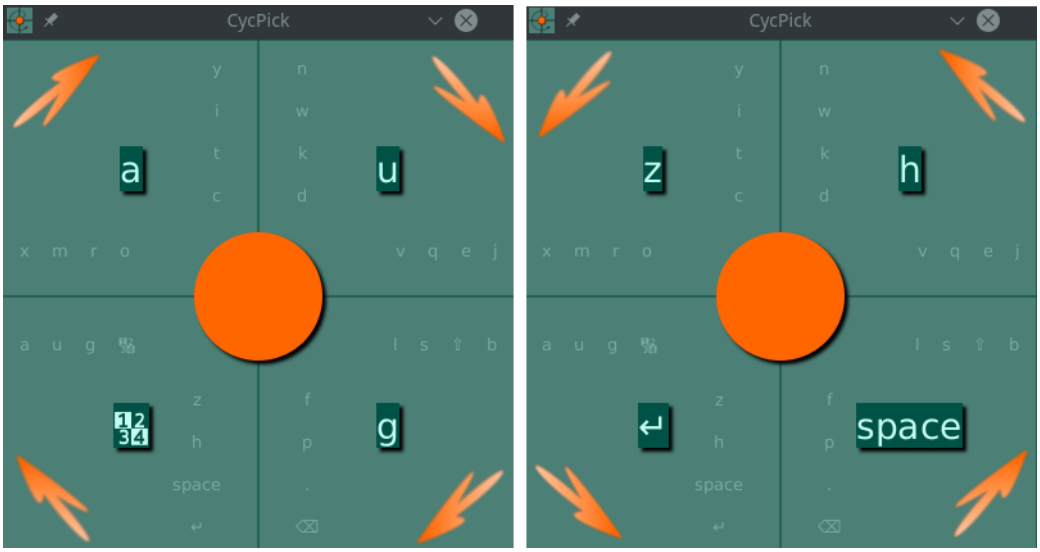
Бирање опције се врши у фази бирања опције која се укључује или искључује левим кликом миша на централно дугме. Након укључивања фазе бирања потребно је ући у један од блокова чиме се започиње одабир. У овом тренутку корисник може изабрати једну од две групе, зависно од тога да ли иде у смеру окретања казаљке на сату или у супротном смеру. Када прође кроз одређену групу, односно кроз ивицу са стране на којој се та група налази онда долази до одабира дате групе, примарни приказ са опцијама постаје мање видљив и приказују се све опције из изабране групе у одговарајућим блоковима, зависно од њихове позиције у групи.



Слика 6: Приказ након избора осме групе опција преласком из четвртог у први блок

Прва опција у групи ће бити приказана у првом блоку, друга у другом и тако даље. У датом примеру (Слика 6) корисник је кренуо активирао фазу бирања, померио миш у трећи блок и након тога одабрао групу проласком кроз заједничку ивицу са првим блоком. Тиме је изабрана осма група. У овом тренутку може да се врати на почетну позицију, чиме ће бити исписано слово “a”, или може да настави у истом смеру и из неког другог блока постави миш назад на почетак чиме би изабрао опцију из датог блока. Овај поступак се врши за сваку опцију, формирајући цикличне покрете приликом бирања, односно писања уколико је реч о текстуалним опцијама. О другим типовима опција биће речи у одељку “Типови и распоред опција”.

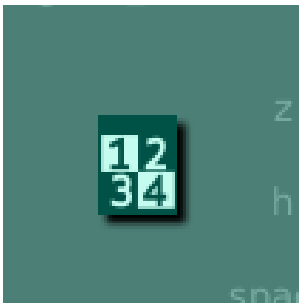
Током бирања се приказују усмерене стрелице у угловима прозора (Слика 7). Оне представљају индикаторе који указују кориснику на смер у коме је потребно вршити бирање, за случај да му је то потребно. Разлог за њихово постојање и релевантност смера при бирању јесто то што од се идеја програма заснива на цикличним поркетима па тиме није могуће мењати смер, започети и завршити бирање у истом блоку (без бирања групе) као ни излазак изван прозора програма током бирања. Ови случаји се сматрају невалидним потезима и при њиховом обављању фаза бирања ће бити искључена чиме се и завршава дато бирање опције.



Слика 7: Приказ стрелица у случају када корисник изабере групу у смеру окретања казаљке на сату (лево) и у случају када изабере групу у супротном смеру (десно)

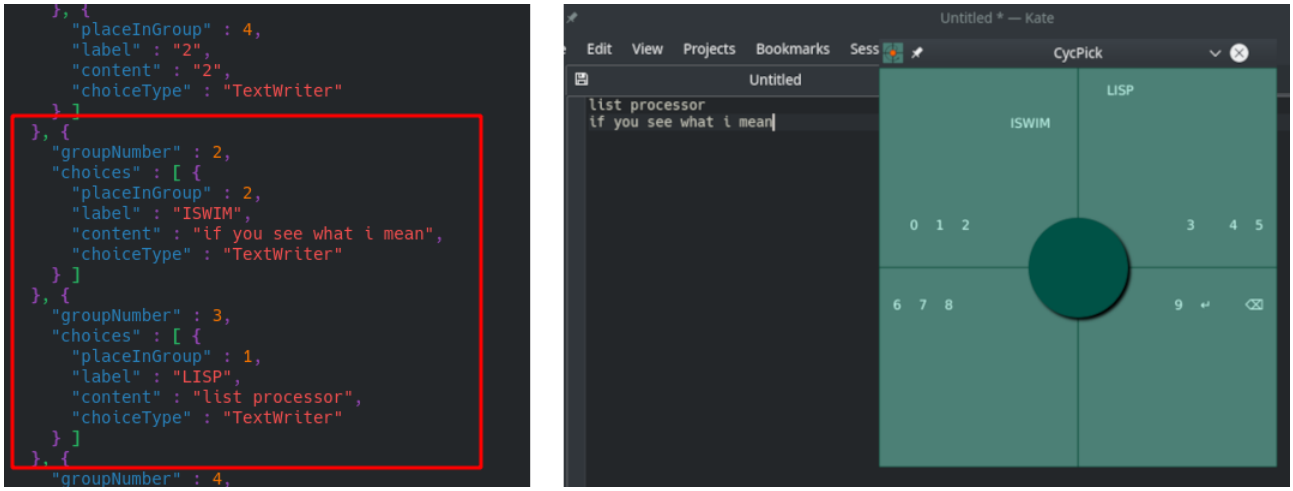
Типови и распоред опција

Програм тренутно подржава два типа опција: **писач текста** (*TextWriter*) и **опцију која садржи подопције** (*ChoiceWithSubchoice*). У претходним примерима коришћен је распоред који садржи само једну опцију типа са подопцијама (Слика 8), док су све остале опције писачи текста. Свака опција, независно од типа, представљена је *Unicode* низом карактера.



Слика 8: Пример опције са подопцијама

Свака опција има садржај који може бити независтан од опције. Тај садржај може бити само један карактер, што би подсећало на дугме на тастатури, а може бити и низ карактера.



Слика 9: Део распореда који се односи на опције са садржајем сачињеним од више карактера (лево) и приказом њиховог коришћења (десно)

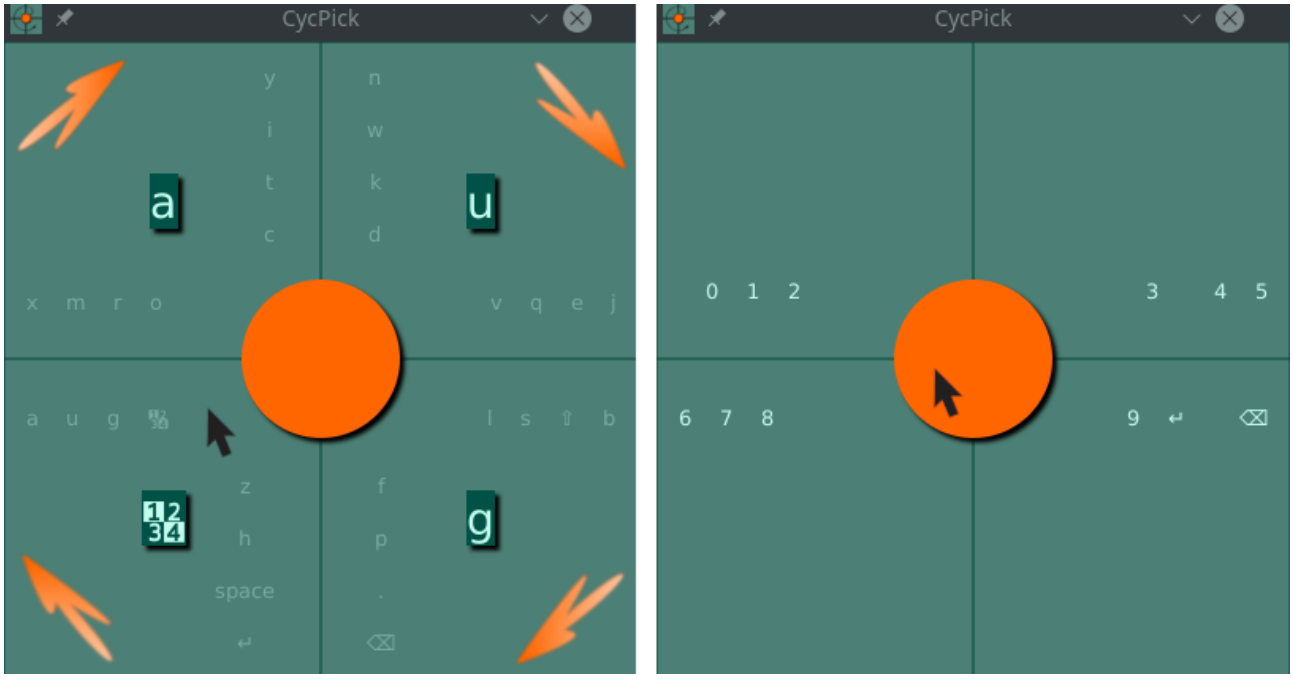
Писач текста

Основни тип опције у програму јесте писач текста. Овај тип омогућава кориснику да неким текстом представи садржај који та опција садржи и да га испише при бирање дате опције. Осим стандардних карактера као што су слова, бројеви и симболи, подржане су и неке опције са посебним значењем као што су *space*, *backspace*, *shift* і *enter*.

Опција која садржи подопције

Поред основног типа опције, постоји и тип опције чијим се одабиром омогућава избор опција које су дефинисане унутар ње. Разлог за постојање овог типа је то што је тренутно број опција у програму ограничен, као и то што корисник може имати потребу за успостављањем хијерархије опција.

Када корисник одабере опцију овог типа доћи ће до промене тренутног распореда опција и уместо њега ће бити приказане подопције изабране опције. Те подопције такође могу бити овог типа чиме се омогућава креирање хијерархије у складу са потребама корисника.



Слика 10: Пример коришћења опције којиа садржи подопције

Уколико корисник жели да се врати назад на претходни распоред опција то може учинити десним кликом миша.

Прилагођавање распореда опцијама

Ради пружања могућности измене редоследа и његовог креирања у складу са потребама корисника, у директоријуму програма креира се фајл *choices.json* који садржи распоред опција. Програм ће користити фајл под овим називом на споменутој локацији. Уколико такав фајл не постоји креира се подразумевани распоред који долази првобитно са програмом.

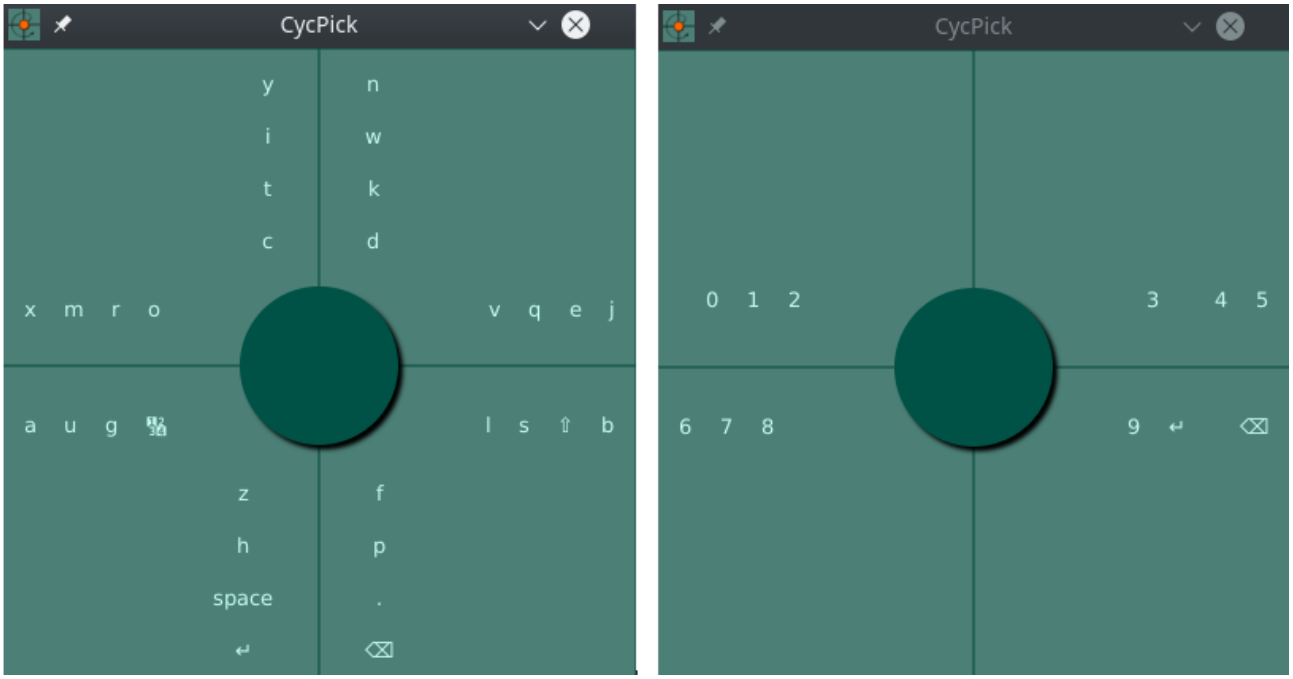
Оно што је битно напоменути је да корисник не мора дефинисати све опције и групе, као у примеру са претходне слике. Уколико се изабере група, односно прође кроз ивицу без дате групе на страни са које се долази, доћи ће до прекида фазе бирања јер се то сматра невалидним избором. Редослед опција и група није битан, већ је дефинисан од стране корисника у одговарајућем пољу.

Слика 11: Део распоред опција из фајла choices.json

Готови шаблони распореда опција

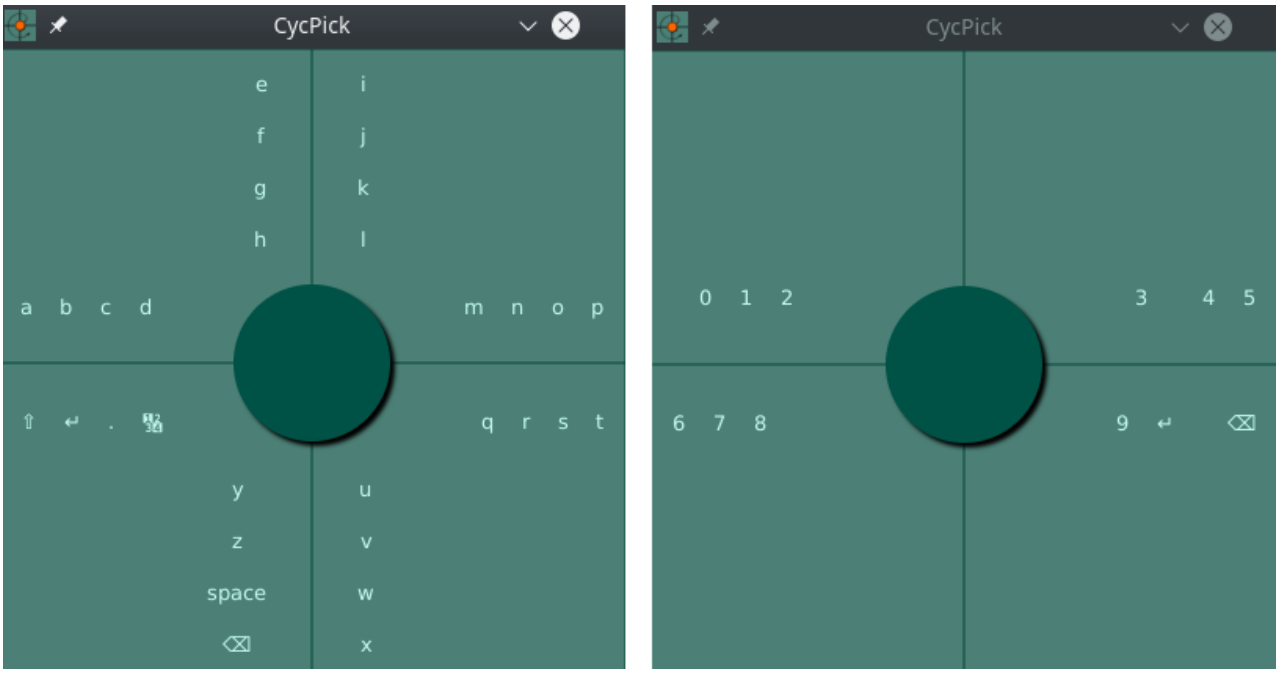
С обзиром на несвакидању природу начина на који се опције бирају, као и њиховог распореда креирани су готови шаблони који би олакшали корисницима прилагођавање распореда кроз пример могућег коришћења, а такође и пружили алтернативу за подразумевани распоред.

Креирана су два шаблона. Први има за циљ да омогући високу ефикасност приликом писања српског и енглеског језика користећи информације о учесталости слова и у једном и у другом језику. Слова која се често користе су распоређена тако да при одабиру групе миш корисника већ буде о одговарајућем блоку чиме би потез завршио само повратком у центар. Слова мање учесталости би била онда распоређена након тих опција, а она слова која имају најмању учесталост би захтевала померање које формира цео круг, потез који најдуже траје. Поред овог начина распоређивања направљено је пар изузетака за опције као што су брисање карактера и додавање белина које су третиране као и чести карактери јер је некада потребно често бирати те опције у низу или их само брзо одабрати.



Слика 12: Шаблон распореда опција за ефикасно писање

Други шаблон има за циљ да олакша проналажење слова коришћењем редоследа енглеске абецеде. Овај распоред омогућава кориснику да на основу одређеног слова претпостави где би тражено слово могло да се налази у односу да то слово.



Слика 13: Шаблон распореда опција за лакше проналажење опција

Исечци изворног кода

```
private def processChoice(choice: Option[Choice]): MoveAttemptOutcome = {
  choice match {
    case None => MoveIgnored
    case Some(c) if c.choiceType == ChoiceType.ChoiceWithSubchoices => {
      currentChoice = choice.getOrElse(rootChoice)
      LayoutChanged
    }
    case Some(c) => {
      c.content match {
        case "space" => {
          robot.keyPress(KeyEvent.VK_SPACE)
          robot.keyRelease(KeyEvent.VK_SPACE)
        }
        case "backspace" => {
          robot.keyPress(KeyEvent.VK_BACK_SPACE)
          robot.keyRelease(KeyEvent.VK_BACK_SPACE)
        }
        case "enter" => {
          robot.keyPress(KeyEvent.VK_ENTER)
          robot.keyRelease(KeyEvent.VK_ENTER)
        }
        case "shift" => shiftPressed = !shiftPressed
        case _ => {
          if (!shiftPressed)
            typeString(c.content)
          else {
            robot.keyPress(KeyEvent.VK_SHIFT)
            typeString(c.content)
            robot.keyRelease(KeyEvent.VK_SHIFT)
            shiftPressed = !shiftPressed
          }
        }
      }

      ChoiceProcessed
    }
  }
}
```

Слика 14: Функција за обраду изабране опције

Овај део кода обрађује изабрани потез. Избор опције која садржи подопције захтева промену тренутног распореда опција на распоред дате опције. Уколико је реч о специјалном типу опције онда се та опција на одговарајући начин обрађују, а уколико је обичан низ карактера у питању онда се само врши испис осим у случају када је изабрана опција *shift*. У том случају се прво симулира притискање дугмета *shift* након чега се исписује изабрана опција и затим се симулира пуштање дугмета *shift.*

```
TodorovicSrdjan
private def convertMoveToChoice(from: Int, to: Int, direction: MoveDirection): Option[Choice] = {
  val groupNumber = findGroupNumber(from, direction)
  val choices = currentChoice.subchoiceGroups.get
    .find(_.groupNumber == groupNumber)
    .map(_.choices)

  choices.flatMap(_.find(_.placeInGroup == to))
}
```

Слика 15: Функција за конвертовање покрета у опцију

Функција *convertMoveToChoice* конвертује потез, односно редни број блока из кога је потез започет, редни број блока у коме се потез завршио и смер започињања потеза у одговарајућу опцију, уколико таква опција постоји. Прво се проналази одговарајућа група за дати блок и смер потеза, а затим се на основу редног броја блока у коме је завршен потез одређује изабрана опција унутар те групе.

```
TodorovicSrdjan
private def ancestorChoice(choice: Choice, startingChoice: Choice = rootChoice): Option[Choice] = {
  choice match {
    case c if c == startingChoice => None
    case c =>
      val choicesOnThisLevel = for {
        groups <- startingChoice.subchoiceGroups
        group <- Option(groups)
        choices <- Option(group.flatMap(_.choices))
      } yield choices

      if (choicesOnThisLevel.get.contains(c))
        Option(startingChoice)
      else {
        var found: Option[Choice] = None
        choicesOnThisLevel.foreach(_.foreach(x=> {
          found =  if found.isEmpty then found else ancestorChoice(c, x)
        }))

        found
      }
  }
}
```

Слика 16: Функција за проналажење претходника одређене опције

Функција *ancestorChoice* представља рекрузивну функцију која тражи претходника дате опције. Почиње од почетне опције (у првој провери је то корена опција) и проверава да ли је реч о почетној опцији. Уколико јесте онда враћа *None* јер у том случају не постоји претходник дате опције. У супротном проверава да ли подопције почетне опције садрже дату опцију и уколико је садрже то би значило да је почетна опција претходна опција па се због тога она враћа позиваоцу. Уколико је почетна не садржи проверавају се подопције подопција почетне

опција. Ако опција налази у некој од њих онда ће се вратити опција која представља претходну опцију прослеђене опције. Ако је не пронађе враћа ње *None*.

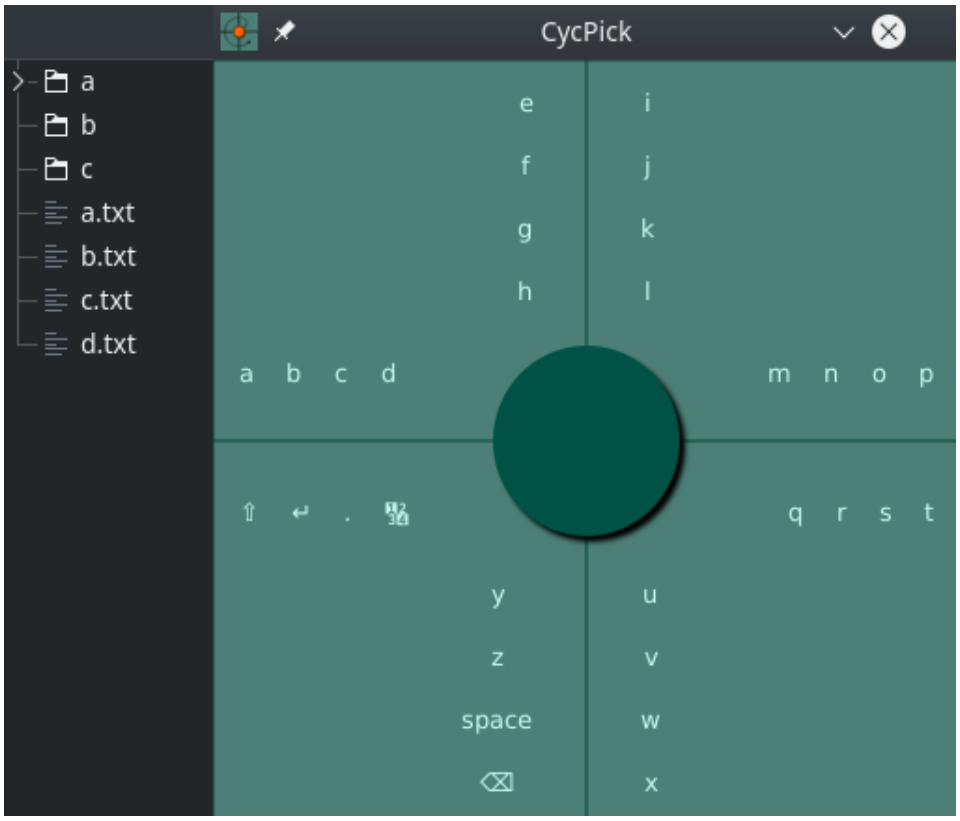
```
TodorovicSrdjan
def choiceLabels(from: Int, direction: MoveDirection): List[String] = {
  val groupNumber = findGroupNumber(from, direction)
  val choices = currentChoice.subchoiceGroups.get
    .find(_.groupNumber == groupNumber)
    .map(_.choices)

  val labels: IndexedSeq[String] = (1 to numOfChoicesPerGroup).flatMap(i =>
    choices.flatMap(_.find(_.placeInGroup == i)
      .map(_.label)
      .orElse(Option(missingChoiceStringRepr))
  ))

  if (labels.nonEmpty)
    labels.toList
  else
    List.fill(numOfChoicesPerGroup)(missingChoiceStringRepr)
}
```

Слика 17: Функција која враћа репрезентација опција одређене групе

Функција *choiceLabels* за редни број блока из кога је започет потез и смер у коме је започет потез враћа листу репрезентација опција из изабране групе која је одређена прослеђеним информацијама. Прво се одређује група на основу прослеђених информација, након тога се проналазе све опције за дату групу. За сваку позицију у групи се проверава да ли постоји опција за њу и уколико не постоји попуњава се репорезентацијом за недостајућу вредност. Уколико се догоди да група не садржи ни једну опцију онда се генерише листа са празним низом карактера.

Могућности даљег развоја

Овај пројекат је имао за циљ да истражи идеју ефикасног рада коришћењем рачунарског миша, слично могућности побољшања ефикасности коришћењем тастатуре коју програми као *Vim* или *Gnu Emacs* пружају. Једна од првобитних идеја јесте била и имплементација више типова опција који би омогућили кориснику да дефинише команде које би желео да се изврше избором дате опције као и навигација кроз фајл систем која би могла да садржи додатан призор са листом фајлова који би били филтрирани на основу одабране опције све док за откуцане карактере се не дође до једног резултата након чега би се вршила његова обрада отварањем или неком другом активношћу (Слика 18).



Слика 18: Пример могућег изгледа претраге фајл система

Једна од потенцијално веома важних идеја за даље истраживање и развој јесте прилагођавање програма за коришћење од стране дизајнера и других људи који већи део времена свог рада користе рачунарски миш, графичку таблу или сличне уредаје. Разлог за то је могућност да се природим покретима руке изврше одређене пречице или напише одређени текст без потреба за коришћењем тастатуре, а дужим коришћењем би се као и код дужег куцања на тастатури побољшала ефикасност при бирању, а тиме и већа ефикасност у раду.

Програм такође може представљати ефикаснију алтернативу за *on-screen* тастатуре па је битно ускладити развој програма и за тај случај коришћења.

Корисни линкови

https://scala-lang.org

https://www.scalafx.org

https://github.com/scalafx

https://docs.oracle.com/javafx/2/css_tutorial/jfxpub-css_tutorial.htm

https://javadoc.io/doc/org.scalafx/scalafx_2.13/latest/index.html

https://github.com/json4s/json4s

https://www.scala-sbt.org/index.html

https://github.com/sbt/sbt-assembly

https://stackoverflow.com/questions/tagged/json4s

https://stackoverflow.com/questions/tagged/scalafx

https://stackoverflow.com/questions/tagged/javafx

https://alvinalexander.com (датум приступа: 16.09.2022)

https://github.com/flide/8VIM (датум приступа: 16.09.2022)